Universidade de Aveiro
2018

Departamento de Eletrónica, Telecomunicações e Informática

JOÃO TIAGO FARIA ALEGRIA

AGREGADOR DE SERVIÇOS DE MOBILIDADE ALTERNATIVA

Universidade de Aveiro
2018

Departamento de Eletrónica, Telecomunicações e Informática

JOÃO TIAGO FARIA ALEGRIA

AGREGADOR DE SERVIÇOS DE MOBILIDADE ALTERNATIVA

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Informática, realizada sob a orientação científica do Professor Doutor Joaquim Arnaldo Carvalho Martins, Professor Catedrático do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, em contexto de estágio em empresa na Ubiwhere Lda., com a supervisão do Mestre João Pedro Pedrosa.

o júri

presidente

Professor Doutor José Manuel Matos Moreira
Professor Auxiliar da Universidade de Aveiro

vogais

Professor Doutor Fernando Joaquim Lopes Moreira
Professor Associado da Universidade Portucalense

Professor Doutor Joaquim Arnaldo Carvalho Martins
Professor Catedrático da Universidade de Aveiro

agradecimentos

Chegar ao final desta etapa, onde concluí o meu percurso académico, vai ser uma etapa que irei sempre lembrar. Nesta, não só aprendi, como cresci e aprendi valores que certamente levarei para futuros capítulos.

O primeiro agradecimento está reservado a quem permitiu que tudo isto fosse possível. À minha família. A minha mãe e o meu pai. O resultado da aposta, confiança e educação torna-me no ser humano que sou hoje.

Em segundo, a todos companheiros desta aventura. De conhecidos a colegas e de conhecidos a amigos. A amizade, o apoio e o companheirismo que levo deles, ao longos destes anos, não só contribuíram para os dias terem um sorriso, mas também, para que as conquistas que fomos somando, fossem mais fáceis.

Todos os professores com quem me cruzei, durante o meu percurso académico, merecem um especial agradecimento, uma vez que contribuíram pela formação que hoje levo.

De seguida gostaria de agradecer ao meu orientador, Professor Doutor Joaquim Arnaldo Carvalho Martins, por todo o apoio e disponibilidade manifestados ao longo da realização deste trabalho.

Gostaria também de agradecer especialmente ao João Pedrosa que se mostrou sempre disponível para me ajudar em qualquer dificuldade, possibilitando assim a conclusão deste trabalho.

Não podia deixar de agradecer também à restante equipa da Ubiwhere que me abraçou durante estes últimos meses. Não sabendo, acabaram por ser a motivação e a força para abraçar mais um dia. Dia a dia. Cada um deles. E para cada um deles, um grande abraço.

"O Homem é do tamanho do seu sonho."
- Fernando Pessoa

palavras-chave

Carsharing, Ridesharing, Táxi, Boleia, Desenvolvimento Web, HTTP, REST, API, SDK, Python, Django, Django Rest Framework, Celery, Crawler, Text Mining

resumo

A crescente preocupação relativa à sobreexploração dos veículos motorizados e dos problemas de mobilidade subjacentes, o congestionamento e o impacto ambiental, despertaram nos últimos anos uma maior sensibilização relativamente à temática.
Desde então, diversas soluções de mobilidade foram apresentadas e têm sido, ao longo do tempo, cada vez mais adotadas, potenciando assim uma diminuição no congestionamento das grandes cidades.
Contudo, o surgimento destas diversas plataformas faz com que o utilizador tenha de despender cada vez mais tempo na procura do melhor serviço para as suas necessidades.

Desta forma, e partindo da proposta lançada pela Ubiwhere Lda., pretende-se com a realização desta dissertação a construção de um agregador que reúna os principais serviços de *carsharing*, *ridesharing*, táxis e semelhantes de forma a reunir numa só solução a possibilidade do utilizador recolher estatísticas de preços, data e hora de partida, duração e distância de uma dada viagem, centralizando assim o processo de procura.

Para o desenvolvimento do agregador apresentado foi utilizado a *framework* Django Rest Framework, que possibilitou a criação da REST API resultante.
Por fim, e de modo a testar a solução final, foi desenvolvida uma prova de conceito que consistiu no desenvolvimento de uma aplicação Android que se conecta, recolhe e apresenta os dados resultantes das consultas à REST API.

Com a criação e desenvolvimento desta solução, a sua integração em projetos futuros, terá como objetivo auxiliar o utilizador na procura de viagens, motivando a adoção das diversas plataformas, levando assim à diminuição do congestionamento sentido nas grandes cidades, tendo como impacto final a diminuição da poluição e a dedução das emissões de poluentes atmosféricos.

keywords

Carsharing, Ridesharing, Taxi, Ride, Web Development, HTTP, REST, API, SDK, Python, Django, Django Rest Framework, Celery, Crawler, Text Mining

abstract

The growing concern about overexploitation of motor vehicles and the underlying mobility problems, congestion and environmental impact, have been in the recent years raised awareness about this topic. Since then, new mobility solutions have been presented and have been increasingly adopted, potentiating a reduction in the congestion of large cities.
However, the emergence of these platforms means that the user has to spend more and more time looking for the best service to satisfy their needs.

In this way, and based on the proposal presented by Ubiwhere Lda., the purpose of this dissertation is the construction of an aggregator that brings together the main services of carsharing, ridesharing, taxis and others, in order to bring together in a single solution, the possibility to collect price statistics, the departure time, duration and distance of a given trip, centralizing the search process.

For the development of the presented aggregator was used the Django Rest Framework, which allowed the creation of the resulting REST API.
Finally, in order to test the final solution, a proof of concept was developed that consisted of the development of an Android application that connects, collects and presents the data resulting from the REST API requests.

With the creation and development of this solution, its integration in future projects, will aim to help the user in the search for travels, motivating the adoption of this kind of platforms, leading to the reduction of congestion felt in the big cities, having as final impact the decrease reduction of emissions of air pollutants.

Conteúdo

Lista de Figuras

Lista de Tabelas

Glossário

Neste documento são utilizadas siglas para referenciar alguns termos cujo significado se encontra descrito de seguida.

API – Application Programming Interface
ASMA – Agregador de Serviços de Mobilidade Alternativa
CRUD – Create, Read, Update, Delete
DOM – Document Object Model
DRF – Django REST Framework
GPS – Global Positioning System
HTML – Hypertext Markup Language
HTTP – Hypertext Transfer Protocol
JDK – Java Development Kit
JSON – JavaScript Object Notation
MTV – Model-View-Template
MVC – Model-View-Controller
ORM – Object-Relational Mapping
PIP – Pip Installs Packages
PoC – *Proof of Concept*
REST – Representational State Transfer
SDK – Software Development Kit
SGBD – Sistemas de Gestão de Bases de Dados
SOA – Service Oriented Architecture
SOAP – Simple Object Access Protocol
SSL – Secure Sockets Layer
SUV – Sport Utility Vehicle
UI – User Interface
URL – Uniform Resource Locator
UUID – Universally Unique Identifier
UX – User Experience

CAPÍTULO **1**

Introdução

1.1. Contexto e Motivação

Atualmente, parte das adversidades sentidas nas grandes cidades dos países desenvolvidos, como é o caso das capitais europeias, são o congestionamento do tráfego e os problemas de mobilidade a si associados. Grande parte desse congestionamento, advém da gestão ineficiente dos meios de transporte particulares, onde, na sua generalidade, o único passageiro acaba por ser o proprietário.

Ao tomar em consideração esta realidade e multiplicando pelo número de utilizadores que exercem esta prática, está diretamente relacionado o aumento dos custos associados ao transporte de pessoas e de bens, bem como os graves impactos ambientais que contribuem para uma intensificação da poluição atmosférica.

Como resposta a estes problemas, verificou-se nos últimos anos, uma alteração dos comportamentos e padrões de mobilidade, com o aumento de práticas baseadas na partilha de boleias, como o *carpooling* ou *ridesharing*, bem como os serviços de *carsharing*. Esta alteração comportamental foi principalmente potenciada pelo crescimento das redes sociais e das aplicações móveis desenvolvidas para os *smartphones*. A primeira facilita e dinamiza as relações interpessoais, permitindo o estabelecimento de múltiplos canais de comunicação entre comunidades, e a segunda, a conveniência de solicitar um serviço de transporte, onde quer que o utilizador se encontre.

Desta forma, os serviços de *carsharing* e de *ridesharing* têm como principal objetivo, contribuir significativamente para o estabelecimento de padrões de mobilidade sustentáveis, criando uma cultura de mobilidade, baseada no uso de várias opções de transporte, ao invés do uso de um veículo pessoal. A ideologia que lhes é subjacente, como o próprio nome indica, visa uma maior consciencialização na partilha de automóveis, de modo a tornar estes serviços mais eficientes [1].

Foi precisamente neste sentido, e com o objetivo de facilitar e promover esta prática, que ao longo destes últimos anos foram desenvolvidas inúmeras plataformas de *ridesharing* e *carsharing*, na sua maioria disponíveis através de aplicações móveis e de forma gratuita, de modo a possibilitar a partilha de boleias entre utilizadores, bem como serviços de aluguer de viaturas, para além da solicitação de táxis ou de serviços concorrentes.

É também particularmente interessante avaliar a presença destes temas num número crescente de artigos científicos a nível mundial, onde o número de ocorrências é cada vez maior. De acordo com o Scopus[1] , o portal *online* com a maior base de dados de textos científicos, o número de publicações relativas à temática de *ridesharing* e *carsharing*, tem tido um grande aumento na última década, atingindo em 2017 as 334 publicações, e em Novembro de 2018, 321 publicações.

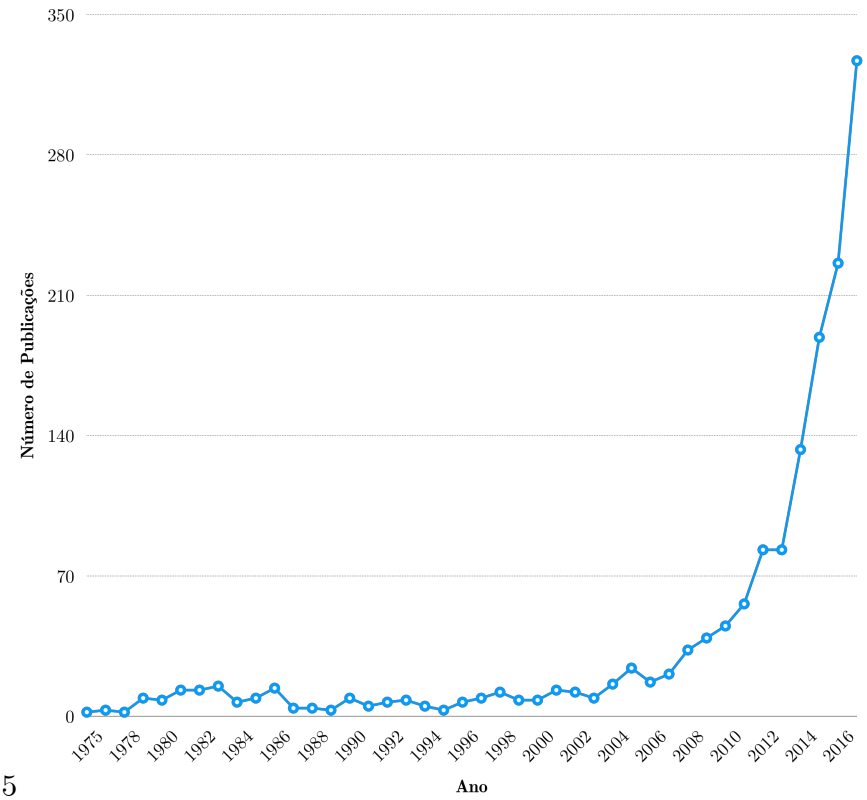


Figura 1 – Comparação do número de publicações por ano com as palavras "ridesharing" ou "carsharing", no título, abstract ou keywords, de acordo com o Scopus, realizada em Novembro de 2018
adaptado: https://goo.gl/iC1uDz

[1] https://www.scopus.com

Por outro lado, a poluição, sobretudo o aumento dos gases de efeito de estufa, tem sido, desde o início do século, a principal preocupação dos ambientalistas, indústrias e entidades governamentais. Associado a este crescimento, fruto da sobre-exploração dos recursos existentes, são cada vez mais exploradas soluções para a diminuição da emissão de gases. Destas, é possível apontar o *carsharing* e o *ridesharing* e, através do mesmo portal, avaliar a existência de artigos relativos ao estudo da melhoria ambiental, que o conceito das viagens partilhadas possivelmente oferece.

Assim, combinando os conceitos da diminuição de poluentes atmosféricos e da melhoria da qualidade do ar, é possível constatar que no mesmo intervalo de anos da pesquisa anterior, houve um aumento considerável de publicações registadas no Scopus.

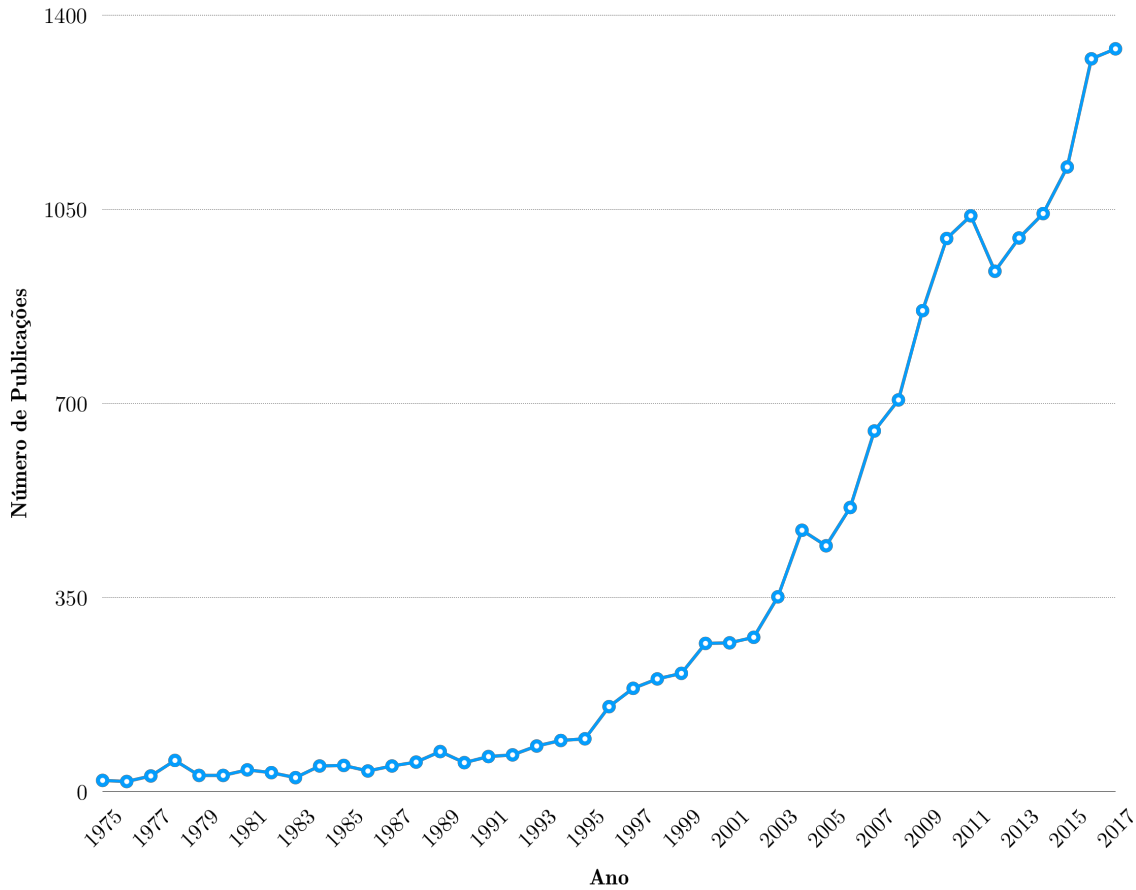


Figura 2 – Comparação do número de publicações por ano com as palavras "atmospheric pollutants", "reduction", "decrease", "air quality", "improvement" no título, abstract ou keywords, de acordo com o Scopus, realizada em Novembro de 2018
adaptado: https://goo.gl/ctDfcz

Desta forma, é possível coligar estas duas pesquisas e acreditar que, com o aumento da partilha de viagens, seja possível reduzir consideravelmente a emissão de gases de efeito de estufa. Ainda que não haja estudos devidamente fundamentados, uma vez que as práticas de partilha de veículos são recentes, é notável que com a diminuição da frota automóvel, existe uma menor emissão de gases poluentes.

Outros estudos, estes relativos aos transportes, colocam o custo anual do congestionamento em 160 mil milhões de dólares [2], onde é estimado um gasto de 960 dólares por condutor, o que implica 7 mil milhões de horas perdidas durante o trânsito e mais de 3 mil milhões de depósitos de combustível queimados. Uma maneira de melhorar o tráfego, é então através da partilha de viagens, onde um estudo conduzido pelo MIT[2] [3], sugere que a adesão ao conceito da partilha de boleias, como a Uber[3] e Lyft[4], poderia reduzir o número de veículos na estrada em um terço.

1.2. Objetivos

O objetivo desta dissertação surgiu como uma proposta lançada pela Ubiwhere[5], uma empresa de tecnologia, sediada em Aveiro e fundada em 2007, onde parte da sua área de atuação se dedica à criação de soluções aplicadas às cidades inteligentes – as *smart cities*. O desenvolvimento destas soluções tem como principal propósito fornecer, a um maior número de indivíduos, condições de sustentabilidade e uma melhoria na qualidade de vida urbana.

É nesse seguimento que assenta o desenvolvimento da solução proposta, onde o principal objetivo, consiste na elaboração de um sistema de informação e respetiva *Application Programming Interface* – API –, seguindo o estilo REST.

[2] http://www.mit.edu
[3] https://www.uber.com
[4] https://www.lyft.com
[5] https://www.ubiwhere.com

Nesta, estarão agregados os mais variados provedores de serviços, tanto de *carsharing* como de *ridesharing*, onde será possível a recolha de estatísticas e informações, como preço e duração da viagem, relativamente a uma dada origem e destino. Também é objetivo a recolha das ofertas de viagens partilhadas através de redes sociais, como é o caso do Facebook, a partir dos grupos existentes na plataforma com esse propósito. Com isto, e integrando o resultado final deste projeto em soluções terceiras, pretende-se oferecer uma diminuição na comparação entre as várias soluções existentes, centralizando o processo de procura de uma viagem numa única solução.

Por fim, pretende-se ainda apresentar uma prova de conceito, sendo esta, o desenvolvimento de uma aplicação Android que tire proveito do sistema e da REST API desenvolvida, não esquecendo de elaborar toda a documentação para uma clara perceção da solução.

CAPÍTULO **2**

Estado da Arte

O número de pessoas que tem vindo a adotar soluções alternativas ao uso do transporte particular, geralmente quando é apenas o único a usufruir da viagem, tem vindo a aumentar nos últimos anos. Na sequência dessa prática, o número de utilizadores dos serviços de *carsharing* e *ridesharing* tem crescido significativamente.

Estes tipos de mobilidade estão, ainda, fortemente associadas a um público mais jovem, pois para estes, o conceito de automóvel deixa de ser encarado como um bem, mas sim como um serviço que permite a deslocação sem os custos inerentes à posse de uma viatura própria, e como a individualização e comodidade face a transportes coletivos, usando o carro apenas quando necessário [4].

Apesar disso, o número de "adultos mais velhos" a utilizar estes serviços, tem vindo a aumentar. Contudo, e segundo um estudo realizado pela Rashmi Payyanadan, estes ainda se mostram um pouco reticentes com estas novas práticas, mostrando preferência em viajar, preferencialmente, com amigos ou familiares, transmitindo-lhes assim, uma maior sensação de segurança e redução nos problemas de comunicação com o condutor, de modo a transmitir as informações relativas ao destino pretendido [5].

2.1. Tipos de Mobilidade Alternativa

2.1.1. Ridesharing

O conceito de *ridesharing* pode ser formalmente definido como uma opção de transporte onde dois ou mais indivíduos, que viajam para o mesmo destino, ou por uma direção semelhante, partilham o custo da viagem, podendo esta ser num táxi, autocarro ou automóvel [5].

Mais recentemente, o conceito de *ridesharing* tem estado associado à rentabilização do veículo pessoal, onde são partilhados os lugares livres com outros passageiros, de uma forma *ad-hoc*.

A partilha *ad-hoc* tem como propósito, a marcação regular de viagens por meios mais informais [6], o diálogo entre os interessados, e ainda a divulgação em plataformas dedicadas à partilha de viagens, ou até mesmo através de publicações em grupos de redes sociais.

Associado a esta ideologia de economia partilhada, é notável, como apresentado anteriormente, uma poupança nos custos de deslocação, nos custos relativos à manutenção de veículos, a redução do volume de tráfego nas cidades e a redução de emissores poluentes associado à utilização singular de um veículo.

De notar que, o conceito de *ridesharing*, apresenta uma longa história com vários métodos de gestão. Para que a partilha seja bem-sucedida, é necessária uma gestão dos itinerários, que incluam a especificação de um lugar de embarque e de desembarque, de um ou mais passageiros.

Desta forma, e segundo a publicação de Furuhata [7], é possível categorizar os diferentes tipos de viagens em dois sistemas:

A primeira, a partilha de viagens não organizadas, envolve sobretudo membros familiares, colegas de trabalho ou amigos próximos. Mesmo sem uma relação pessoal próxima, é possível proceder-se à partilha de viagens, como é o caso do pedido de uma boleia. Contudo, este tipo de partilha não é o mais escalável devido aos meios de comunicação limitados e ineficientes.

Por outro lado, a partilha de viagens organizadas, está sob a responsabilidade de companhias que oferecem uma plataforma que suporta a partilha de viagens, onde é possível a consulta e a reserva por parte de um interessado. Desta forma, a possibilidade de consulta e reserva é uma das características fundamentais que estes tipos de provedores oferecem, ao contrário do que acontece com um táxi ou boleia, que são tipicamente solicitados nas ruas.

É ainda possível classificar o sistema de partilha de *ridesharing* em quatro padrões, conforme ilustrado na Figura 3. Os padrões apresentados de seguida, são descritos para o caso de um único passageiro; porém, os mesmos podem ser aplicados a múltiplos passageiros. Deste modo, a notação utilizada denota a como o condutor e b como o passageiro, onde ambos têm a sua origem o e o seu destino d.

É também denotado *u* como o local de levantamento e *v* como o local de entrega, sabendo que originalmente o condutor tem a sua rota, *R(a)*, e a viagem partilhada, formada entre o condutor e o conjunto de passageiros, é denotada como *R(a,B)*.

- Padrão 1 – Viagem Idêntica – Tanto a origem como o destino do condutor a e do passageiro b são idênticos, isto é, $o_a = o_b = u_b \wedge d_a = d_b = v_b$

- Padrão 2 – Viagem Inclusiva – Tanto a origem o_b e o destino d_b do passageiro b, compõe a rota original do condutor a, R(a), ou seja, $o_b, d_b \in R(a)$. Resumidamente, o destino pretendido pelo passageiro não é idêntico ao do motorista, mas faz parte da rota deste último.

- Padrão 3 – Viagem Parcial – Tanto o local de levantamento, u_b, como o local de entrega, v_b, do passageiro b, fazem parte da rota do condutor a, $R(a)$, mas nem a origem nem o destino do passageiro, pertencem à rota do condutor, ou seja, $u_b, v_b \in R(a) \wedge \neg(o_b = u_b \wedge d_b = v_b)$. A partilha da viagem apenas será a viagem de b.

- Padrão 4 – Viagem com Desvio – O local de levantamento, u_b, ou o local de entrega, v_b, ou ambos, relativamente ao passageiro b, não fazem parte da rota original do condutor a, $R(a)$. Assim, ao realizar um desvio, a viagem $R(a, B)$ abrange tanto o local de levantamento como o de entrega. Além disso, é ainda possível distinguir esta em dois sub-casos:
  1) Ambos os locais de levantamento e entrega coincidem com a origem e destino do passageiro, ou seja, $o_b = u_b \wedge d_b = v_b$
  2) Ou o caso contrário, os locais de levantamento e entrega não coincidem com a origem nem destino do passageiro, $\neg(o_b = u_b \wedge d_b = v_b)$

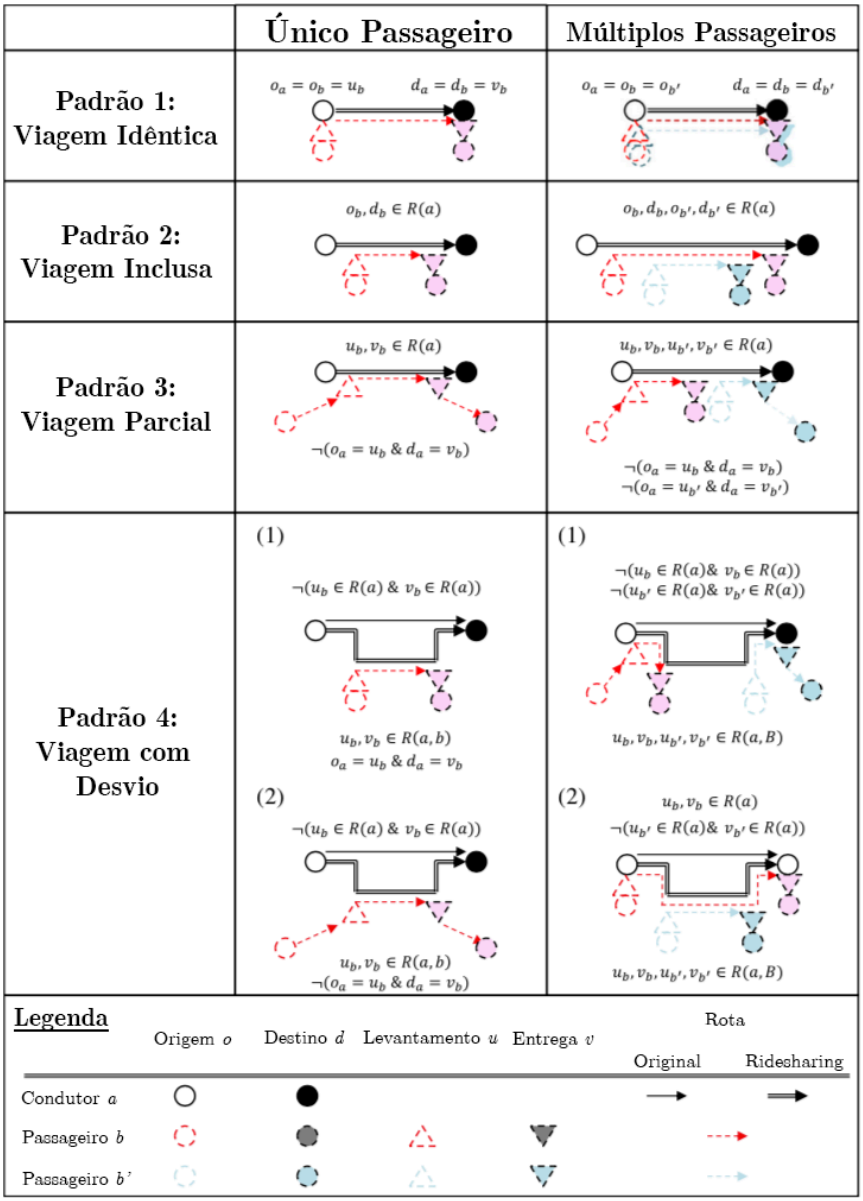


Figura 3 – Esquematização dos padrões de ridesharing adaptado: The state-of-the-art and future directions (Furuhata, et al., 2013)

Contudo, nem todos os padrões são aplicados quando a partilha da viagem está à responsabilidade de uma agência competente. Segundo o mesmo estudo, a correspondência das viagens entre o condutor e o passageiro, é tanto maior quanto a proximidade do local de levantamento do passageiro, em relação ao local de partida do condutor. Também é referido que a partilha de uma viagem com desvio, ou seja, o padrão 4 apresentado anteriormente, pode causar reticências na tomada de decisão para o condutor que está a oferecer, já que os custos adicionais da viagem e o tempo de desvio, não costumam ser suportados financeiramente, pelo passageiro.

2.1.2. Carsharing

O conceito de *carsharing* é apresentado como um sistema de aluguer de viaturas, onde um ou mais indivíduos têm acesso a um veículo, normalmente num intervalo de tempo reduzido, onde é partilhada uma viagem, gerida e assegurada por uma organização terceira [8]. Segundo Shasheen, a primeira prática de *carsharing* ocorreu no pelo final da década de 1940, com o objetivo de possibilitar um maior acesso a veículos, apresentando um custo de deslocação mais baixo [9].

A comodidade e a oferta apresentadas pelas companhias responsáveis pelo aluguer das viaturas foram fatores que beneficiaram o crescimento desta prática. Refere-se que em inícios de 2015, esta prática contava com cerca de 1.5 milhões de utilizadores, unicamente na América do Norte. Nos dias de hoje, o *carsharing* é uma prática disponível em várias áreas urbanas em todo mundo, sendo o número de utilizadores cada vez maior [10].

Contudo, convém referir que o serviço de *carsharing* não deve ser confundido com os serviços de *rent-a-car*, onde as diferenças entre ambos são bem significativas. Normalmente, o aluguer de viaturas por parte das companhias de *rent-a-car* é taxado ao dia, o que não acontece nos serviços de *carsharing*, onde os veículos podem ser alugados por horas ou até mesmo por minutos. Também nos serviços de *carsharing*, as reservas, o levantamento e a devolução, são realizados diretamente pelo utilizador, sem quaisquer intermediários, podendo estas ser realizadas em diferentes locais de uma cidade, onde o utilizador escolhe e reserva o veículo que se encontra mais perto de si e que satisfaça, de igual modo, as suas necessidades. Geralmente, este processo está otimizado para aplicações destinadas ao mercado *mobile*, onde é necessário um registo prévio no serviço e um comprovativo de habilidade de condução [4].

Com isto, é ainda possível classificar o sistema de partilha *carsharing* em dois tipos. No primeiro, o *round-trip*, os veículos são levantados e devolvidos no mesmo local e, no segundo, o *one-way*, os utilizadores têm a liberdade de devolver o veículo requisitado numa localização diferente face ao local de levantamento.

O sistema *round-trip* era até ao final da década passada, o sistema mais comum e utilizado; porém, e com o crescimento do número de veículos elétricos, dos *smartphones* e dos sistemas de informação, houve um crescimento significativo na adoção do *one-way*.

Este último, o sistema *one-way*, pode ainda ser classificado em dois tipos:

O primeiro, o *station-based*, é caracterizado pelo aluguer de veículos, num intervalo de tempo reduzido, onde é permitido o levantamento do veículo num posto e a sua devolução, noutro. As vantagens destes serviços baseiam-se na confiança e na segurança de onde os veículos serão estacionados, bem como a capacidade de os reservar antecipadamente. No entanto, isto pode reduzir a liberdade do utilizador e comprometer parte do itinerário definido pelo mesmo.

Por outro lado, no serviço *free-floating*, o foco é também o aluguer de veículos ao minuto, tendo a particularidade de ser possível estacionar o veículo em qualquer das zonas permitidas, ou seja, nas áreas em que o serviço atua. Este tipo de partilha é mais atrativo, financeiramente, para viagens curtas, dando ainda maior flexibilidade ao utilizador. Outra particularidade deste serviço, é a possibilidade de verificar em tempo real a disponibilidade dos veículos e reservá-los por meio de um *smartphone* ou qualquer outro dispositivo conectado à internet. Em certas companhias, é ainda possível, através do *smartphone*, o desbloqueio e o bloqueio do veículo, por forma a indicar o início e término do serviço de aluguer.

Apesar destes tipos de sistemas serem bastante inovadores, existem ainda desafios a serem superados, como é o exemplo da gestão do estacionamento nas garagens dos postos de aluguer. De notar que, os problemas de estacionamento associados ao sistema *free-floating*, podem ser reduzidos, uma vez que o veículo pode ser estacionado em qualquer das ruas autorizadas, diminuindo o congestionamento das garagens, comparativamente ao serviço *station-based*. Contudo, ao diminuir o número de lugares em parques de estacionamento, estará a ser comprometido o estacionamento em zonas públicas, reduzindo a oferta de lugares de estacionamento aos demais utilizadores.

Na tabela seguinte encontram-se, com intenção de sumarização, os diferentes sistemas de *carsharing*, explanados anteriormente.

Tabela 1 – Classificação dos sistemas de carsharing

	Tipos de *Carsharing*		
	Round-Trip	*One-way*	
		Station-based	*Free-Floating*
Levantamento do veículo	Em determinados postos	Em qualquer estação de aluguer de viaturas	Em qualquer sítio que esteja disponível
Entrega do veículo	Mesmo local de levantamento		Em qualquer sítio que seja autorizado
Problemas de estacionamento	Não	Sim	Sim / Não

adaptado: A Critical Analysis of Travel Demand Estimation for New One-Way Carsharing Systems (Vosooghi, et al., 2017)

Desta forma, e com a adoção desta e da anterior prática, são esperados benefícios do *carsharing* quer financeiros quer ambientais, relacionados com a propriedade de um veículo particular, como detalhado em 2.3 – Avaliação do Impacto Causado.

Os primeiros estudos, demonstram empiricamente essa promessa, conforme detalhado na Tabela 2, onde foram avaliados os diferentes sistemas de *carsharing*. Os dados apresentam estudos realizados em dois continentes – Europa e América – onde a experiência de utilização dos serviços de *carsharing* é elevada, e estes devem ser vistos como indicadores qualitativos. De notar que, na maioria dessas cidades onde o serviço era oferecido, inúmeros utilizadores reduziram a propriedade de veículos através do uso de veículos de *carsharing*.

Tabela 2 – Sumarização da redução relativa à propriedade de um veículo próprio, em relação aos utilizadores de serviços de carsharing

Região / País	**Ano da Pesquisa**	**Sistema de *Carsharing***	**Redução da propriedade do veículo**
EUA (Portland)	1999	*Round-Trip*	Entre os entrevistados, 26% venderam o seu veículo pessoal e 53% renunciaram a compra de um carro
América do Norte (várias cidades)	2004	*Round-Trip*	Cerca de 20% dos utilizadores dos serviços de *carsharing*, reduziram a propriedades de carros particulares. Um veículo *carsharing*, substitui entre cinco a seis carros particulares.
	2008	*Round-Trip*	A propriedade média de um veículo reduziu de 0.47, na razão veículo/domicílio, para 0.24. Um veículo *carsharing*, removeu de 4 para 6, o número de veículos particulares nas estradas.
Europa (várias cidades)	2009	*Round-Trip*	Percentagem de utilizadores que abandonaram o uso do carro privado: Bélgica: 15.7%, Suíça: 31.6%, Alemanha: 16%
Reino Unido (Inglaterra e País de Gales, excl. Londres)	2014 a 2015	*Round-Trip*	Um veículo de *carsharing*, removeu 4 veículos particulares das estradas e adiou a compra de mais de 9 carros
Reino Unido (Londres)	2014 a 2015	*Round-Trip*	Um veículo de *carsharing*, removeu 8.6 veículos particulares das estradas e adiou a compra de mais de 19,8 carros
América do Norte (várias cidades)	2014 a 2015	*Free-Floating*	Um veículo de *carsharing*, removeu 1 a 3 veículos particulares da estrada e adiou a compra de 4 a 9 carros.

adaptado: Vehicle ownership reduction - A comparison of one-way and two-way carsharing systems (Namazu, et al., 2018)

2.2. Recetividade em Portugal

Em Portugal, o primeiro serviço de *carsharing* surgiu em Setembro de 2008 em Lisboa durante a Semana Europeia da Mobilidade. No início de 2009, 12 veículos estavam disponíveis para a utilização de cerca de 100 clientes privados e comerciais. O primeiro serviço de partilha de carros de Portugal foi organizado pela Carristur[6], uma subsidiária da operadora de transportes públicos de Lisboa, responsável pela a organização de serviços para turistas, bem como de serviços regulares de transporte público [1]. Posteriormente a esta iniciativa, Portugal começou a receber as primeiras soluções em larga escala deste tipo de serviços, salientando o grande impacto e influência do Uber, como o maior serviço concorrente à atividade taxista, e o BlaBlaCar[7], como o "número um" das atividades de *ridesharing* na Europa. A apresentação de alguns dos serviços existentes e a missão que cada companhia de partilha de veículos pratica, está disponível para consulta em 2.4 – Apresentação e análise de soluções existentes.

Mais recentemente, no ano de 2015, foi aprovado em Conselho de Ministros e publicado em Diário da República, na Resolução do Conselho de Ministros n.º 54/2015, o ECO.mob – Programa de Mobilidade Sustentável para a Administração Pública 2015-2020, que promove a redução do número de viaturas, bem como os gases que contribuem para o efeito de estufa, impulsionando assim a adoção de energias renováveis. O ECO.mob assenta em três grandes eixos de atuação, sendo esses, a gestão da mobilidade, tecnologia e comportamentos. O seu principal objetivo é assim melhorar a eficiência económica e desempenho ambiental das deslocações associadas à Administração Pública. Com a implementação do ECO.mob, prevê-se uma redução significativa dos impactos ambientais, bem como uma poupança de 50 milhões de euros, entre os anos de 2015 a 2020 [11].

[6] https://www.carristur.pt
[7] https://www.blablacar.pt

2.3. Avaliação do impacto causado

Com a adoção destes novos serviços de mobilidade, estão associados vários impactos, tanto para o utilizador final, como para o meio ambiente. Relativamente aos impactos ambientais, é possível indicar a redução da frota de veículos em circulação, bem como a diminuição da poluição e a redução das emissões de poluentes atmosféricos, como é o caso do dióxido de carbono, CO_2, e do metano, CH_4 [12], conduzindo assim a uma melhoria da qualidade do ar, como apresentado no estudo do capítulo anterior.

No que diz respeito aos impactos sociais, é possível apontar uma poupança ao utilizador relativamente a combustíveis, seguros e outros encargos associados à propriedade de um automóvel, a possibilidade de transporte por pessoas que não detenham um veículo próprio, bem como a escolha oportuna de um destes, através da utilização das aplicações de aluguer e partilha de carros, existentes no mercado *mobile* [13].

No segmento de lazer, como é o caso dos festivais de verão, a adoção dos serviços de *ridesharing* tem sido uma tendência crescente, sobretudo nos festivais que decorrem na Europa, aumentando a notoriedade e reputação destes. Baseado na partilha dos assentos livres com outros passageiros, que tenham o mesmo percurso que o condutor, esta filosofia contribui para a redução do custo das viagens, bem como, no número de lugares de estacionamento, que se pode apresentar escassa neste tipo de eventos. O serviço da "boleia partilhada" apresenta melhores resultados em festivais, onde a acessibilidade através de transportes públicos é limitada, e a adoção deste acontece, sobretudo, através de grupos específicos em redes sociais ou através de plataformas de *ridesharing*, que apresentam uma maior garantia tanto para o condutor como para o passageiro [14].

Relativamente ao impacto causado no turismo, e sendo Portugal considerado o melhor destino turístico do mundo em 2017 [15], a adoção dos serviços de *carsharing* e *ridesharing*, bem como dos serviços alternativos aos táxis, proporciona uma maior oferta de mobilidade e de comodidade aos turistas, uma vez que a solicitação dos serviços pode ser realizada através das aplicações móveis.

Outro aspeto positivo é o método de pagamento que estes serviços adotam. O uso do cartão de crédito, bem como o Paypal[8], são facilidades que os turistas, ao desfrutarem destes serviços, irão sentir uma vez que, são métodos de pagamento além-fronteiras.

Por último, e não menos importante, o impacto económico pode ser particularmente interessante, uma vez que pode ser inserido em tipos de mercado bastante específicos, também conhecidos como nichos de mercado. Um exemplo prático é a adoção deste tipo de serviços por parte das empresas, ao invés de possuírem um veículo próprio para serviços ocasionais [13].

2.4. Apresentação e análise de soluções existentes

De modo a conhecer parte das ofertas atualmente existentes, os tópicos seguintes apresentam algumas das soluções de *carsharing* e *ridesharing* já desenvolvidas e disponíveis para os entusiastas destas práticas de mobilidade. Parte destas plataformas, apresentadas de seguida, serão as que integrarão a solução agregadora proposta no desenvolvimento da dissertação.

2.4.1. Uber

A Uber é, possivelmente, a empresa de maior renome à escala mundial na área de mobilidade alternativa. É uma empresa de tecnologia, com o principal propósito de fornecer diferentes tipos de transporte aos seus utilizadores, estando presentes em mais de 750 cidades, com milhões de passageiros a cada dia [16]. Fundada em Junho de 2009 [17], a empresa atua em Portugal, desde 2014, sobretudo nas três cidades mais turistas, sendo elas, Lisboa, Porto e Algarve [18].

Para os seus utilizadores, os passageiros do serviço, a Uber é essencialmente um novo sinónimo de um táxi. É sobretudo através das aplicações móveis desenvolvidas pela mesma, disponíveis nos principais sistemas operativos móveis como iOS, Android e Windows Phone, que é possível conectar tanto motoristas como utilizadores, através de recursos como GPS.

[8] https://www.paypal.com/pt

Assim, é possível saber a localização de ambas as entidades, afim de efetivar o processo de reserva do serviço e de conhecer o tempo previsto que este demora até se apresentar junto do(s) passageiro(s) [19].

Para efetivar a solicitação de uma viagem, o utilizador necessita da aplicação instalada no seu *smartphone*, bem como de uma conta previamente criada. De seguida, e depois de escolher o seu destino, é enviado um pedido de viagem para os condutores mais próximos do utilizador. O motorista, que estará a conduzir o seu veículo pessoal, deslocar-se-á até ao utilizador e levá-lo-á ao destino solicitado.

Relativamente ao cálculo do trajeto, este é conseguido através do Waze[9], uma comunidade de utilizadores que, através do uso da aplicação pelo *smartphone*, permite de forma colaborativa reportar, em tempo real, situações que obstruam a rápida circulação, como grandes filas de espera ou até possíveis acidentes [20], permitindo assim definir, em tempo real, qual a rota mais rápida até ao destino pretendido [21].

A própria aplicação móvel, oferecida pela Uber, responsabiliza-se pelo cálculo automático da rota para o condutor, distância do ponto de partida até ao ponto de chegada e a tarifa que o(s) passageiro(s) terá(ão) de pagar [22].

Referentemente aos métodos de pagamento, é tanto possível realizar o pagamento através do serviço de Paypal, do pagamento com o cartão de débito e de crédito, ou ainda em monetário, sabendo que este último está cada vez menos a ser adotado, e que podem diferir consoante o país e cidade em que opera [23] [24]. Após o término da viagem, é efetuado o pagamento conforme o método escolhido, e o comprovativo de pagamento será enviado para o e-mail, previamente indicado no registo do utilizador.

[9] https://www.waze.com/pt-PT

Serviços Uber

A oferta de serviços abrangida pela Uber, direciona-se desde o passageiro citadino, a grupos de turistas ou até a serviços executivos de limusine. A título exemplar, é possível apresentar o UberX e o Uber Green, como serviços que disponibilizam a mobilidade até 4 passageiros, e que apresentam uma tarifa a metade do custo de um táxi que opera na mesma cidade.

Existem outros serviços disponíveis, cada um destinado a satisfazer um público alvo distinto, desde serviços que consigam transportar mais passageiros, até serviços de nível executivo com direito a limusine de luxo. [25] [26]

Produtos Uber

O grande crescimento que o Uber apresenta, é também devido ao número de produtos e soluções que vai oferecendo, sobretudo aos seus clientes [27].

A principal atividade, e a que proporcionou o crescimento da empresa, foi o pedido de viagens com a oferta de novos meios de mobilidade, concebidos sobretudo com o propósito de auxiliar os passageiros a chegarem aos seus destinos, de uma forma económica. Para o cálculo do custo da viagem, é tomada em consideração uma tarifa base; uma tarifa para o tempo e distância estimados na rota; uma percentagem face à procura de viagens para a área desejada – também conhecida como tarifa dinâmica, ou *multiplier* –; uma taxa de reserva e ainda eventuais portagens associadas [28]. A tarifa dinâmica pode ser entendida como uma taxa, quando o nível de procura é elevado, havendo assim, mais procura de viagens do que motoristas disponíveis. Com isto, é espectável que haja ocasiões onde o preço possa atingir valores que ultrapassem o dobro do pré-estabelecido, o que se pode tornar não suportável para moradores em áreas turísticas, e que sejam adeptos deste tipo de mobilidade [29].

Mais recentemente, em 2014, a Uber Technologies Inc., começou a apostar no mercado de serviços de entrega, mais especificamente, na entrega de refeições. Desta forma, com o Uber Eats[10], é possível encontrar e encomendar as refeições que cada cliente mais aprecia, do leque de restaurantes presentes na plataforma.

[10] https://www.ubereats.com

Semelhantemente à aplicação de viagens, o pagamento do transporte da refeição pode ser efetuado eletronicamente e pode também ser possível o acompanhamento do pedido, em tempo real e pelos mapas da aplicação, desde o restaurante, até ao sítio especificado para a entrega [30].

Já no setor empresarial, e no ano de 2017, a companhia apresentou uma solução idêntica à experiência de transporte de passageiros mas, desta vez, aplicado ao transporte de cargas e de mercadorias – o Uber Freight[11], sendo assim possível o deslocamento de bens de uma cidade para outra [31].

Integração dos Serviços Uber no ASMA

A API desenvolvida e disponibilizada publicamente pela própria Uber, oferece aos desenvolvedores, a habilidade de recolher informações relativas às viagens disponíveis de momento, e integrar esses recursos em aplicações de terceiros. Para tal, é necessário a especificação de um local de origem e destino, de modo a reunir estatísticas associadas à viagem, como o exemplo dos preços aplicados e o cálculo do tempo necessário até um condutor se apresentar ao utilizador [32].

Na verdade, a ferramenta disponível para os desenvolvedores, é um SDK [33], uma solução mais abstrata e organizada, comparativamente a uma API, onde é possível aglomerar um conjunto de APIs numa única solução e executar assim as ações pretendidas. As chamadas a este serviço, podem assim solicitar informações relativas aos tipos de carros disponíveis, localização do condutor, expressa em coordenadas geográficas, estimativas do tempo e do preço (incluindo a conversão da moeda, se aplicável), bem como o histórico e atividade de conta do utilizador e até reservar uma viagem [34].

Deste modo, é possível criar diferentes tipos de soluções, como aplicações móveis ou *websites*, que reúnam as informações previamente recolhidas.

[11] https://www.uberfreight.com

2.4.2. Cabify

Conhecida como uma das principais concorrentes aos serviços de transporte Uber [35], a Cabify[12] é uma empresa fundada e sediada em Espanha, desde 2011, com o propósito de conectar motoristas e passageiros, numa única plataforma [21], de modo a oferecer um serviço de transporte privado a estes últimos. Além de atuar no país onde surgiu, a empresa também está presente em mais 10 países.

Em Portugal, está presente sobretudo em cidades turísticas como o Algarve, Lisboa, Porto e Funchal, mas revela-se com uma forte presença sobretudo nos países da América Latina como Brasil, México, Chile e Colômbia [36].

De modo a solicitar uma viagem, o utilizador necessita de ter na sua posse a aplicação instalada no seu *smartphone*, disponível para iOS e Android e, diferentemente da sua concorrente, a Uber, o Cabify possibilita aos passageiros a solicitação de um transporte a partir de um *browser*, ou seja, sem a necessidade do *smartphone*.

O processo de reserva é bastante semelhante ao apresentado anteriormente, onde, ao indicar o local da partida e de chegada pretendidos, são apresentados em tempo real, os motoristas mais próximos de si, afim de realizar a viagem pretendida. O utilizador aguardará pela confirmação de um destes que deslocar-se e levá-lo-á ao destino solicitado. Assim como no Uber, o cálculo do trajeto é realizado através do Waze, que define, em tempo real, qual a melhor rota a adotar, tomando em consideração possíveis engarrafamentos e acidentes [21].

O próprio serviço é responsável pelo cálculo do valor da rota, antes mesmo do utilizador o confirmar e, um dos principais diferenciais que a Cabify apresenta face à Uber, é que o valor é determinado ponto a ponto, isto é, o custo é calculado pela quilometragem percorrida, independentemente do trânsito, ou caminho alternativo que o motorista possa optar [29].

De um modo geral, isto significaria uma maior economia para o passageiro que, aparentemente, não teria que pagar a mais pelo serviço no caso de escolher um horário ou destino mais congestionado ou desvios na rota.

[12] https://www.cabify.com/pt

Porém, existe uma tarifa mínima que é aplicada e que pode ser diferente consoante a cidade de execução, e os preços praticados tendem a ficar mais baixos, consoante o número de quilómetros percorridos [37].

Relativamente aos métodos de pagamento e comprovativos de pagamentos, a Cabify apresenta os mesmos métodos de pagamento que a sua rival Uber, podendo estes variar ainda consoante a cidade do passageiro.

Serviços CABIFY

Atualmente, os serviços oferecidos pela Cabify são quatro, podendo estes não estar em vigor na totalidade dos países em que esta atua.

Deste modo, podemos apontar o Cabify Lite como um serviço análogo ao UberX; o Cabify Baby, ideal para transportar crianças em segurança, com um custo adicional em Portugal de €4 [38]; o Cabify Express, que permite o transporte de objetos e encomendas, desde pequenos pacotes, documentos ou presentes para alguém [39] e o Cabify Pets, que oferece a possibilidade de transporte de animais de estimação, desde que o mesmo vá dentro da bolsa de transporte, no entanto, cães-guia poderão viajar sem necessitarem de ser acompanhados por bolsas ou caixas de transporte [40].

Integração dos Serviços CABIFY no ASMA

Contrariamente à sua rival, a Cabify não disponibiliza o acesso à sua API sem uma avaliação prévia. Desta forma, quando contactada e explicado o propósito do acesso aos seus serviços, a Cabify acabou por rejeitar, não sendo assim possível a integração dos mesmos na solução final.

2.4.3. Lyft

O terceiro exemplo que visa reformular o conceito dos serviços de táxi, é a Lyft[13]. Sendo esta mais um adversário direto aos serviços prestados pela Uber, a Lyft destaca-se ao apresentar preços mais baixos e por adotar um conceito mais informal, descontraído e descomplicado [41].

Fundada três anos após a sua principal concorrente, em 2012 [42], esta apresenta-se mais como uma solução que aproxima tanto motoristas como utilizadores de serviços de carros partilhados.

Como principal foco no mercado americano, a Lyft presta os seus serviços em aproximadamente 300 cidades dos Estados Unidos [43] e, em Dezembro de 2017, apresentou os mesmos em Toronto, entrando assim no mercado canadiano, com o principal objetivo de competir com a Uber, que já marcava presença no Canadá, expandindo assim os seus serviços [44]. Atualmente, a Lyft ainda não avançou com uma previsão de chegada a Portugal, nem a outro país do mercado Europeu [45].

Semelhantemente às plataformas anteriores, a metodologia de uso da Lyft é idêntica às apresentadas até então. Primeiramente, é necessária a instalação da aplicação para dispositivos móveis, disponível para iOS e Android e a criação de uma conta. Nesta, é necessário o preenchimento de informações relativas à identificação do utilizador, bem como o método de pagamento, oferecendo apenas a disponibilidade de cartão de crédito e de débito.

Seguidamente, e agora apto para usufruir da solução, o utilizador indica o destino pretendido na interface principal que, após solicitar a sua viagem, aparecerão no mapa, os motoristas mais próximos aptos para a execução da viagem. Da oferta apresentada, o utilizador pode selecionar o mais conveniente para si, comparando o tipo de carro, a distância a que o condutor está de si, classificação e a tarifa aplicada pelo mesmo [46].

[13] https://www.lyft.com

Serviços LYFT

Tal como os seus concorrentes, a Lyft oferece diferentes categorias e valores, dos quais: o Lyft (versão *standard*), que se apresenta como um serviço análogo ao UberX e ao Cabify Lite, transportando até 4 passageiros; o Lyft Plus ideal para grupos maiores de até 6 passageiros; o Lyft Line apresentado como uma oferta mais económica, já que adota o conceito de *ridesharing*, uma vez que permite a partilha da viagem com outros utilizadores que escolham o mesmo itinerário e a divisão do custo da viagem pelo número de utilizadores que tenham ingressado a mesma e a Lyft Premier, como serviço de luxo, ideal para transportes empresariais. [41]

Integração dos Serviços Lyft no ASMA

O acesso público aos serviços de desenvolvedor da Lyft é bastante recente. Apenas a meio do desenvolvimento desta dissertação é que a Lyft facultou o acesso às ferramentas de desenvolvimento de um modo análogo ao que a Uber fornece, o que não acontecia anteriormente, onde era necessário elaborar um pedido e poderia ou não ser aprovado pela própria, como aconteceu no início da elaboração da dissertação.

Após o registo nesta plataforma, e de modo a recolher as informações relativas às viagens disponíveis num dado momento, estão disponíveis todas as chaves de acesso na *dashboard* da solução, para que seja possível a conexão futura aos serviços da mesma.

Um dos aspetos positivos da plataforma de desenvolvedores da Lyft é o processo de autenticação, onde o mesmo é realizado através de um *token* enviado para o número de telefone registado no perfil dos mesmos.

As bibliotecas disponíveis de forma oficial apenas dão, atualmente, suporte aos sistemas operativos móveis iOS e Android, porém estão de igual modo, disponíveis outras bibliotecas mantidas por uma comunidade de membros que oferecem suporte a outras linguagens de programação, como é o caso do SDK desenvolvido em Python[14], para soluções que adotem esta linguagem.

[14] https://pypi.org/project/lyft_rides/

2.4.4. BlaBlaCar

Diferentemente como apresentado até este momento, o BlaBlaCar[15] adota o conceito de *ridesharing*, ou seja, partilhar uma viagem e dividir os custos da mesma, consoante o número de passageiros que ingressem nesta [47].

Sediada em França e fundada em 2006, a empresa está atualmente presente em 22 países, incluindo Portugal [48], e o seu propósito é proporcionar uma forma alternativa e mais barata de viajar, combinando assim os lugares livres de um carro com passageiros, que estejam à procura de uma viagem, cujo destino seja próximo do condutor [49].

Esta é uma solução que se apresentou aos portugueses em Outubro de 2012 [48], por forma a apresentar uma oferta diferenciadora e substancialmente mais barata de viajar [47], comparativamente a alguns transportes públicos, como autocarros ou comboios [50]. A publicação e procura de viagens nesta plataforma de *sharing economy*, está tanto disponível através da navegação pelo *browser* ou da aplicação móvel, disponível para iOS e Android [51].

O funcionamento da partilha de viagens, tem duas óticas distintas possíveis de analisar. A primeira, a do condutor, que organiza as viagens, publica o número de lugares livres e especifica o trajeto e o preço por lugar. E a segunda, a do passageiro, que pesquisa as suas viagens, especificando a data, e o seu ponto de partida e de chegada. Seguidamente, este escolhe um dos condutores presentes nos resultados da pesquisa com base na hora e preço da viagem. Após a seleção, o passageiro entra em contacto com o condutor, através de um sistema interno de mensagens ou então telefonicamente, para que juntos, organizem a sua viagem partilhada de forma transparente [52].

Para que tal seja possível, o utilizador necessita de estar registado na plataforma do BlaBlaCar, de modo a contactar um outro membro ou partilhar os lugares livres do seu carro. Todavia, se desejar apenas consultar viagens e preços, o registo não é necessário [53].

[15] https://www.blablacar.pt

O preço da viagem é fixado pelo condutor e não é negociável. Quando este publica os lugares livres para uma dada viagem, o valor é aplicado de igual forma para o número de passageiros que ingressem na viagem. Contudo, e por forma a que o condutor não escolha o preço de forma arbitrária, este baseia-se no preço recomendado pela BlaBlaCar de acordo com o itinerário e os gastos associados, nunca excedendo um máximo definido pela mesma, de modo a assegurar uma boa prática na atribuição do preço da viagem [54].

Os métodos de pagamento possíveis de adotar, conforme explicitados pela BlaBlaCar, são o cartão de crédito ou o Paypal no final da confirmação da viagem e, após o sucesso desta, será enviado um e-mail de confirmação da reserva [55] [56]. No caso de pagamento com Paypal, um recibo de pagamento será enviado pela entidade [57].

No entanto, noutros países, conforme indicado por outras fontes, o pagamento pode ser realizado presencialmente com o condutor no final da viagem, sem passar pela plataforma [58] [59].

Integração dos Serviços BLABLACAR no ASMA

A API da BlaBlaCar, permite aos desenvolvedores a procura de viagens e a extração de todas as informações associadas à mesma, para que seja possível a integração destas em soluções terceiras.

Para tal, o primeiro passo é a criação de uma conta de desenvolvedor, que lhe permitirá obter a chave de autenticação da API, essencial para estabelecer a conexão à mesma. A conexão é assim realizada através de pedidos HTTP e os formatos de retorno suportados são o XML ou JSON.

Os tipos de informações possíveis de retorno são a consulta de viagens e os detalhes das mesmas onde, para resultados mais específicos, é necessário construir uma *query* mais precisa. Assim, para além da data, do local de partida e de chegada – que são campos imprescindíveis para a consulta de uma viagem –, é possível indicar, por exemplo, o número de acentos livres ou um montante máximo a pagar.

Uma das limitações possíveis de apontar nesta, é a restrição a 10.000 consultas diárias, o que não se tornará escalável com a publicação desta solução no mercado. A solução apresentada pela BlaBlaCar, de modo a cessar esta limitação, é o

contacto direto com os mesmos, de modo a adquirir uma licença por forma a obter um número ilimitado de consultas [60].

2.4.5. 99

A plataforma apresentada em seguida é a 99[16], anteriormente designada como 99Taxis [61], que surgiu sobretudo como uma forma de sustentabilização do mercado tradicional de táxis, numa perspetiva de acompanhar os concorrentes, cuja finalidade se centra na oferta de transportes alternativos ao uso do carro privado e de serviços de *ridesharing*.

Esta aplicação, desenvolvida e publicada em 2012 [62], apresenta-se como uma resolução ao problema que a maioria das pessoas enfrenta quando deseja solicitar um serviço de táxi – não possui o número de um taxista ou da central mais próxima, ou até mesmo não dispor de saldo suficiente para a realização da chamada [63].

Sendo este o maior serviço de táxis do Brasil, a média de viagens realizadas por mês é aproximadamente de 1 milhão, com cerca de 110 mil táxis e 5 milhões de utilizadores registados. Em Portugal, esta solução surgiu no final de 2014, primeiramente em Lisboa – como a primeira cidade europeia a receber o serviço – e de seguida no Porto, onde apresenta mais de 800 táxis e 10 mil utilizadores registados, de acordo com dados recolhidos em 2015.

Assim, e para que um utilizador possa solicitar um táxi à distância de poucos cliques, o procedimento é praticamente análogo face às aplicações já apresentadas anteriormente, sendo assim primariamente necessária a obtenção da aplicação móvel, disponível para iOS, Android e Windows Phone. De seguida, o sistema encontrará as viaturas mais próximas, onde o utilizador aguardará pela confirmação por parte de um taxista.

Dos benefícios apresentados pela 99 é possível evidenciar que este processo é completamente realizado sem a intervenção de uma central, facilitando a comunicação entre o passageiro e o taxista. Após a confirmação do taxista, o nome, foto e número de telemóvel, aparecerão na aplicação, de modo a permitir um fácil contacto entre ambos.

[16] https://99app.com

Relativamente ao sistema de pagamento, a 99 oferece a possibilidade de pagamento *in-app* através de Paypal e cartão de crédito, ou diretamente com o taxista, através do pagamento em numerário, cartão de débito ou crédito [64].

Como referido anteriormente, a aplicação 99 visa fortalecer o mercado tradicional dos táxis e, de acordo com Pedro Fonseca, responsável pela 99 no mercado nacional, "*A aplicação tornou-se numa ferramenta essencial para o dia-a-dia do taxista, que poderá não só oferecer um serviço de vanguarda como aceder a mais clientes. Além disso, traz uma nova experiência ao utilizador pela agilidade, segurança e qualidade do serviço prestado. O crescimento de 30% por mês [em 2015] do número de viagens realizadas em Portugal, confirma a forte adesão ao 99Taxis que, no curto prazo, será a mais importante aplicação mobile de táxis também em Portugal*" [64].

Serviços 99

De um modo semelhante ao que tem sido apresentado, os serviços que a 99 apresenta são diversos, apresentando-se como uma solução completa de mobilidade urbana, oferecendo diferentes serviços para diferentes situações.

Assim, é possível apontar [65]: o 99Taxi, como o serviço de táxis da plataforma; o 99POP, como o serviço de *ridesharing* da 99 com foco numa economia partilhada e na partilha de lugares livres de um automóvel com outros passageiros; e a 99TOP/LUXO, destinado para viagens longas e para clientes que procuram conforto.

Integração dos Serviços 99 no ASMA

Da mesma forma que a Cabify, não foi possível o acesso à plataforma de desenvolvedores da 99. Aquando contactada e explicado o propósito do acesso aos seus serviços, a 99 acabou por rejeitar, não sendo assim possível a integração dos mesmos na solução final.

2.4.6. MyTaxi

O MyTaxi[17] apresenta-se como um serviço semelhante ao da 99 – uma modernização do serviço tradicional de táxis, permitindo criar uma ligação direta entre o taxista e o cliente – que concorre diretamente com as outras aplicações de táxis existentes, mas também com outros serviços como a Uber e Cabify.

Fundada em Junho de 2009, 3 meses antes da Uber [66], a MyTaxi foi a primeira aplicação móvel para táxis a nível mundial e, desde então, tem vindo a estender os seus serviços por toda a Europa e Estados Unidos da América. Atualmente, está presente em mais de 40 cidades e conta com uma média de 45 mil táxis registados e com mais de 10 milhões de descargas nas lojas móveis, estando deste modo presente nas plataformas iOS e Android [67].

É através destas que o utilizador realiza a solicitação de um táxi, onde é localizada a posição atual do mesmo e envia para os taxistas mais próximos o pedido. Com isto, evita-se que os taxistas andem pelas cidades à procura de clientes, o que acarreta melhorias ao nível do tráfego das cidades [66] e a possibilidade de solicitar um táxi sem intervir com centrais de táxis.

É ainda possível adicionar parâmetros que o utilizador ache oportunos para a viagem, como por exemplo, um veículo com mais lugares, possibilidade de transporte de animais, especificar um dado motorista ou escolher a categoria do táxi, como um de gama alta ou um táxi ecológico. Na eventualidade do utilizador não especificar um dado condutor, a atribuição destes não é arbitrária, mas sim com base na pontuação atribuída por outros passageiros. Esta informação está disponível para consulta, aquando da seleção de um motorista, e é possível apresentar o nome, fotografia e contacto – onde é permitido contactá-lo para o informar de alguma alteração ou fornecer mais indicações – do mesmo, bem como a marca, modelo e a matrícula da viatura [68].

[17] https://pt.mytaxi.com

Relativamente ao método de pagamento, é possível pagar a viagem através de cartão de cartão de débito, cartão de crédito, Paypal ou em dinheiro, diretamente ao motorista [69] e a estimativa da tarifa, bem como o tempo estimado da viagem são calculados e apresentados ao utilizador, antes de efetivar a reserva [68].

Comparativamente aos seus concorrentes, a MyTaxi tem como diferencial, a possibilidade de fazer reservas antecipadas até quatro dias antes da data, ou oferecer gorjeta ao taxista [66].

2.4.7. Thumbeo

O Thumbeo[18] apresenta-se como uma aplicação *mobile* de "boleias contextualizadas", desenvolvida por dois antigos alunos da Universidade de Aveiro, em parceria com duas empresas da região de Aveiro, a Ubiwhere e Oakreative[19], sendo eles João Pedro Pedrosa, um dos orientadores desta dissertação, e Eduardo Duarte. Desenvolvida em 2014 [70], e lançada oficialmente em 2015, com a ideia inicial de criar um sistema semelhante ao já apresentado pela Uber, rapidamente alteraram a estratégia, acrescentando a natureza "contextualizada", como a aplicação se apresenta. É através do suporte à pesquisa com o uso de *hashtags*, também conhecidas como palavras-chave ou termos de elevado interesse, que o utilizador consegue realizar pesquisas sobre assuntos da sua preferência, como é o caso de festivais e de eventos de grande renome, onde se consegue apontar o fecho de 35 parcerias, até ao momento da elaboração desta dissertação [71]. Ainda relativamente ao uso das *hashtags*, o cliente pode subscrever o serviço de notificações associadas a um determinado termo que este demonstre interesse, sempre que sejam criadas viagens com esse destino.

Oferecendo a possibilidade do utilizador ser um condutor e um passageiro numa única aplicação, é possível criar, quer boleias pontuais, como um agregado de boleias, com a oferta de ida e volta, quer agendar boleias semanais [72]. Deste modo, se um passageiro desejar também ser um condutor, apenas terá de registar um veículo na área pessoal da aplicação.

[18] http://www.thumbeo.com
[19] http://www.oakreative.com

Assim, e depois de registado, o utilizador terá acesso a um mapa com todas as boleias disponíveis à sua volta, origem, destino e rota, num raio de 50 quilómetros. Pode também consultar o *rating* dos condutores, os lugares disponíveis ou a data e hora da boleia com origem e destino.

Ao submeter um pedido de boleia, o proprietário da viatura é notificado e este poderá aceitar ou recusar o mesmo. O utilizador, independente da opção do condutor, será notificado com a decisão deste permitindo assim receber *feedback* relativamente à sua submissão.

O Thumbeo integra também um serviço de mensagens, onde é possível a comunicação com todos os intervenientes da viagem, viabilizando, por exemplo, um ponto de encontro entre condutor e passageiro.

Relativamente ao preço das viagens e a método de pagamento destas, o primeiro é definido pelo condutor e o pagamento poderá ser realizado em numerário ou por outra via acordada entre passageiro e condutor.

Associado ao crescimento, qualidade e notoriedade, em 2016, a aplicação recebeu a nomeação para a categoria *Best use of Technology pela Iberian Festival Awards* [73].

À data de 2017, mais de 95% dos utilizadores registados na aplicação eram portugueses. Contudo, também já contam com utilizadores de países como Inglaterra, Espanha, França, Itália, México e EUA [70].

Integração dos Serviços THUMBEO no ASMA

A integração dos serviços do Thumbeo não foram possíveis de serem integrados na solução uma vez que a mesma se encontrava, em manutenção, aquando o desenvolvimento desta dissertação.

2.4.8. CityDrive

Com o crescimento da população que tende a desistir do seu carro particular, sobretudo nas grandes cidades metropolitanas, o CityDrive[20] é uma empresa de *carsharing* com maior histórico em Portugal, fundada em 2014 [74], cujo principal foco é o aluguer de carros ao minuto, através de um dispositivo conectado à internet.

O serviço pode ser usado tanto pelo acesso a um *browser*, como pelo uso das aplicações *mobile*, disponíveis para iOS e Android [75]. O primeiro passo para o uso deste serviço é a realização do registo.

Neste, serão pedidos dados para a concretização do mesmo, bem como o *upload* de uma fotografia da carta de condução e do cartão de cidadão, de modo a validar a habilidade de condução por parte do utilizador.

Após registado na plataforma, o utilizador recebe a localização dos veículos mais próximos de si, onde está indicada a distância até ao mesmo, a morada, a matrícula, o modelo, a percentagem de combustível que o carro tem no depósito e o preço, dando assim oportunidade de reservar o carro mais conveniente. Porém, se o utilizador desejar consultar todos os carros disponíveis, existe uma opção para esse fim, onde é possível consultar o mapa da cidade com a indicação de onde se encontram todos os carros CityDrive, desde que não estejam a ser utilizados.

Depois da reserva de um automóvel, é iniciado um período de 15 minutos de *stand-by* [74]. Neste período, o carro está reservado, desligado, mas ao dispor do cliente [4], como uma estimativa do tempo que o utilizador demorará a apresentar-se junto da viatura. No entanto, também é possível reservar o carro quando se está perto deste, evitando assim o tempo de *stand-by*.

O passo seguinte que se coloca é o de abrir o carro. Não é necessário levantar a chave em nenhum ponto de encontro, pois esta encontra-se no interior do carro

[20] http://www.citydrive.pt

e à vista de qualquer um! A abertura do carro apenas acontece através da aplicação móvel sendo que, no prazo máximo de 30 segundos, se abrem.

Uma vez desbloqueado, as chaves encontram-se normalmente perto do rádio, devidamente seguras. Para as usar, é necessário selecionar a opção "ativar o veículo" na aplicação, já que o carro não dá à ignição caso não seja ativado pelo utilizador. Este é um serviço de segurança equipado nos carros da CityDrive, permitindo que, mesmo alguém entre no carro e tente utilizar a chave, não o irá conseguir.

Quando o utilizador termina a sua viagem, basta estacionar numa das zonas permitidas, colocar a chave junto do sítio em que a encontrou, sair do carro, voltar à aplicação e carregar na opção "Terminar Processo".

O grande inconveniente possível a apontar é, de facto, o estacionamento, sendo este apenas possível no centro de Lisboa (zona verde), incluindo os lugares geridos pela EMEL – Empresa Municipal de Mobilidade e Estacionamento de Lisboa. Na zona amarela da cidade, os carros podem permanecer estacionados durante 12 horas após terminar a viagem, podendo ser acrescida uma taxa de 10€ ao último condutor, caso o veículo permaneça por mais de 12 horas sem voltar à zona verde. As restantes zonas do país são de zona azul, onde se pode circular com o carro, porém a viagem só termina e o processamento do pagamento é realizado, quando o veículo regressar à zona verde.

O combustível está incluído no serviço e os custos variam consoante o número de horas em que a viatura permaneceu alugada, sabendo que não são cobrados encargos adicionais como seguros, combustível, estacionamento ou mensalidades [4] [74].

Integração dos Serviços CityDrive no ASMA

Atualmente, o CityDrive não dispõe de um serviço dedicado para desenvolvedores e métodos de comunicação para a recolha de informações para integrar no agregador.

2.4.9. DriveNow

Foi através da Brisa[21], uma operadora de infraestruturas de transportes em Portugal, que Lisboa foi a primeira cidade da Península Ibérica a receber os serviços *carsharing* em sistema *free-floating* da DriveNow[22], uma empresa cujo principal foco é a partilha de carros. Inicialmente, o serviço arrancou com 211 viaturas, das quais 11 elétricas. Atualmente, a DriveNow está presente em mais 12 cidades europeias e tem cerca de um milhão de clientes registados, que usufruem do serviço diretamente da *app mobile*, disponível para iOS e Android.

Para um utilizador usar a aplicação, terá de efetuar o registo na mesma, o qual tem um custo de €10, que servirá de crédito na aplicação como forma de pagamento. O funcionamento que esta adota é muito semelhante ao previamente descrito na CityDrive, ou seja, os condutores que usufruam deste serviço, após chegarem ao destino pretendido, apenas têm que estacionar, desde que seja numa zona verde da área de Lisboa [4], e concluir o processo pela aplicação. O aluguer dos veículos também é pago ao minuto, e neste estão incluídos seguro, combustível e estacionamentos, livres de qualquer taxa mensal. [76] [77]

Integração dos Serviços DriveNow no ASMA

Atualmente, o DriveNow não dispõe de um serviço dedicado para desenvolvedores e métodos de comunicação para a recolha de informações para integrar no agregador.

[21] http://www.brisa.pt

[22] https://www.drive-now.com/pt/pt/lisbon

2.4.10. BookingDrive

O serviço de *carsharing* da BookingDrive[23] está disponível desde 1 de Dezembro de 2016 em Portugal e, diferentemente das propostas anteriores, esta plataforma é um serviço *online* de aluguer de carros entre particulares, permitindo assim aos proprietários das mesmas obterem um rendimento extra no final de cada mês, ao alugar o seu veículo.

Desenvolvido inteiramente em Portugal, o sistema abrange todo o país e conta com sensivelmente 1700 utilizadores e cerca de 70 viaturas registadas. O registo, bem como a utilização do serviço é feito através do *browser*, não apresentando assim suporte para nativo para equipamentos móveis. Contudo, o BookingDrive apresenta, no seu *site*, uma forma de adicionar à tela inicial do smartphone um acesso rápido para o serviço, também conhecido como *WebApp*. [4] [78] [79]

Ao utilizar este serviço, a BookingDrive possibilita a rentabilidade dos veículos nos momentos em que não são usados, permitindo a todos os particulares o acesso a uma extensa oferta de veículos a um preço mais acessível. O registo da viatura terá de ser feito pelo dono da mesma, e apenas só serão aceites viaturas até 11 anos, tendo o dono da viatura de garantir a inspeção periódica, seguro e imposto de circulação em dia.

O aluguer tem assim uma duração mínima de 2 horas e um máximo de 7 dias, podendo estes ser renovados [78]. O pagamento do serviço é apenas realizável unicamente através de cartão de crédito [80] e o preço definido pelo proprietário [81], onde este e o condutor combinam, através de um *chat* disponível na plataforma, o local de encontro e as demais informações relativas ao aluguer. De notar que, tanto na entrega como na devolução do veículo, é assinado um contrato de aluguer, de modo a garantir o bom estado do mesmo [82].

[23] https://www.bookingdrive.com

Integração dos Serviços BookingDrive no ASMA

Atualmente, o BookingDrive não dispõe de um serviço dedicado para desenvolvedores e métodos de comunicação para a recolha de informações para integrar no agregador.

2.4.11. TaxiFareFinder

O TaxiFareFinder[24] é uma solução *online* que permite a consulta dos preços e das tarifas aplicadas aos serviços de táxi, em mais de 1000 cidades ao redor do mundo. Segundo apresentado, as estimativas oferecidas pelo TaxiFareFinder são mais precisas do que qualquer outro serviço comparador de táxis, disponível *online*.

As estimativas são resultado de um algoritmo desenvolvido pelos próprios, onde são tomadas em consideração uma série de aspetos, ao invés de apenas a distância e duração da viagem, mas também aspetos como os padrões de tráfego de uma dada cidade, velocidade, densidade urbana e possíveis tempos de espera [83].

Integração dos Serviços TaxiFareFinder no ASMA

Após vários meses de negociação, o acesso a um *access-token* que permitisse a conexão e recolha de informação relativas a serviços de táxis foi concebido. A API do TaxiFareFinder dispõe de vários *endpoints* que permitem a consulta de várias estatísticas e informações, como por exemplo, encontrar uma cidade suportada pelo TaxiFareFinder através de uma coordenada; obter estatísticas, como a distância, duração e preço de uma viagem e ainda quais as companhias taxistas que operam numa dada cidade [84].

Contrariamente às outras soluções já apresentadas, o TaxiFareFinder não dispõe de uma área reservada a desenvolvedores. O processo de solicitação, discussão e entrega do *acess-token* foi através da troca de *e-mails*.

[24] https://www.taxifarefinder.com

Tabela 3 – Sumarização das soluções apresentadas

	Opera em Portugal	**Aplicação Móvel**	**API**		**SDK**	
			Possui	**Acesso facultado**	**Possui**	**Acesso Facultado**
Uber	√	√	×	-	√	√
Cabify	√	√	√	×	×	-
Lytf	×	√	√	√	Não Oficialmente	√
BlaBlaCar	√	√	√	√	×	-
99	√	√	√	×	×	-
MyTaxi	√	√	√	×	×	-
Thumbeo	√	Apenas Android	√	×	×	-
CityDrive	√	×	×	-	×	-
DriveNow	√	√	×	-	×	-
BookingDrive	√	×	×	-	×	-
TaxiFareFinder	√	×	√	√	×	-

2.5. Análise de plataformas agregadoras de mobilidade

O crescimento de soluções que oferecem alternativas à utilização do veículo privado, ou a oportunidade de partilhar viagens com outros utilizadores, têm vindo a multiplicar-se ano após ano. Apesar de grande parte destes operar apenas em certas cidades, os provedores deste tipo de serviços têm vindo a expandir e a oferecer maior cobertura dos seus serviços nos países em que operam.

Relativamente às soluções existentes, e que tenham uma finalidade idêntica à solução final desta dissertação, extrair e apresentar as informações relativas às viagens oferecidas pelos diferentes *providers*, estas têm surgido maioritariamente nos últimos dois/três anos. Porém, e após consultadas as principais lojas de aplicações móveis (Apple App Store[25] e Google Play[26]), um vasto número de soluções apenas se dedica à comparação das duas grandes americanas – a Uber e o Lyft –, ou apenas oferece soluções para determinadas zonas.

Desta forma, as plataformas apresentadas de seguida são as que, numa ótica pessoal, aglomeram um maior número de soluções, tanto para sistemas de *carsharing* como de *ridesharing*, tornando-se assim mais convenientes para atender às necessidades de um dado utilizador.

2.5.1. GoogleMaps

Um dos primeiros serviços a incluir a oferta de estimativas de preços e de tempo de chegada de vários provedores de serviços de *ridesharing*, foi a aplicação de mapas da Google[27], em meados de 2016. Apesar de um lançamento controverso, uma vez que a Uber, até então, não permitia a integração dos resultados da sua API em ferramentas de comparação de preços, a adição destes acabaria por beneficiar ambas as empresas. Aquando do lançamento, foram introduzidos na aplicação os dois maiores fornecedores – Uber e Lyft– aumentando a oferta destes ao longo dos meses. [85]

[25] https://www.apple.com/pt/ios/app-store/

[26] https://play.google.com

[27] https://www.google.com/maps

A vantagem que o Google Maps oferece é, sem dúvida, uma interface de utilizador bastante completa e familiar, para os utilizadores já experientes em outras soluções Google. Nesta, estão incluídas várias alternativas de transporte e os seus trajetos, sendo essas, o automóvel, os transportes públicos, o trajeto pedestre e a integração de serviços de *ridesharing*. Relativamente à oferta dos serviços de *ridesharing*, a integração destes na aplicação, faculta as principais informações necessárias aquando da consulta e reserva de uma viagem, sendo estas a estimativa de custo da viagem, de chegada e a duração da mesma.

O processo de reserva da viagem já transcende o propósito da aplicação, contudo, para cada viagem selecionada, existe um botão que direciona o utilizador a concluir a solicitação da viagem e o pagamento da mesma, na aplicação do provedor.

Um exemplo da utilização atual deste serviço na aplicação do Google Maps é apresentado de seguida.

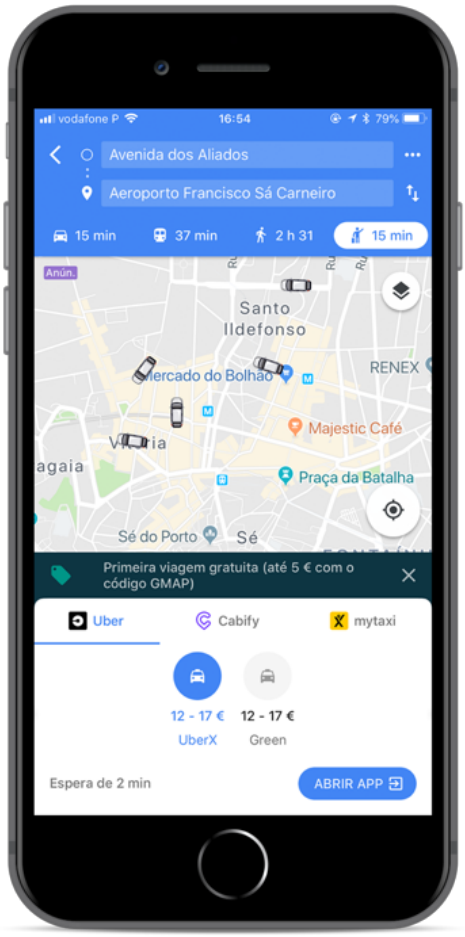


Figura 4 – Exemplo de Utilização do Google Maps com provedores de serviços de ridesharing

2.5.2. Free2Move

A Free2Move[28] apresenta-se como uma empresa de aluguer de veículos elétricos e de veículos não motorizados, agregando dezenas de soluções, assentando-se numa mobilidade sustentável. Marcando presença sobretudo na Europa e nos Estados Unidos da América, o principal destaque da marca é o serviço de *carsharing*, com o aluguer ao minuto das viaturas que disponibiliza.

A *interface* da aplicação apresenta, como grande parte das soluções de transporte, as ofertas disponíveis num mapa alusivo à área em que opera. Um dos benefícios disponíveis na aplicação, é a possibilidade de delinear, geograficamente, a área de atuação de cada *provider* que ofereça serviços para determinada área, como observável no recorte à esquerda da Figura 5. Outra funcionalidade disponível, é a apresentação dos resultados no formato de listagem; contudo numa ótica pessoal enquanto utilizador, esta apresenta uma desvantagem, pelo facto de não incluir técnicas de segmentação de resultados, quando estes são em grande número. Dessa forma, podia estar presente a ordenação das viagens pelo custo, proximidade do veículo, quantidade de combustível na viatura ou seleção de um determinador *provider*.

[28] https://www.free2move.com

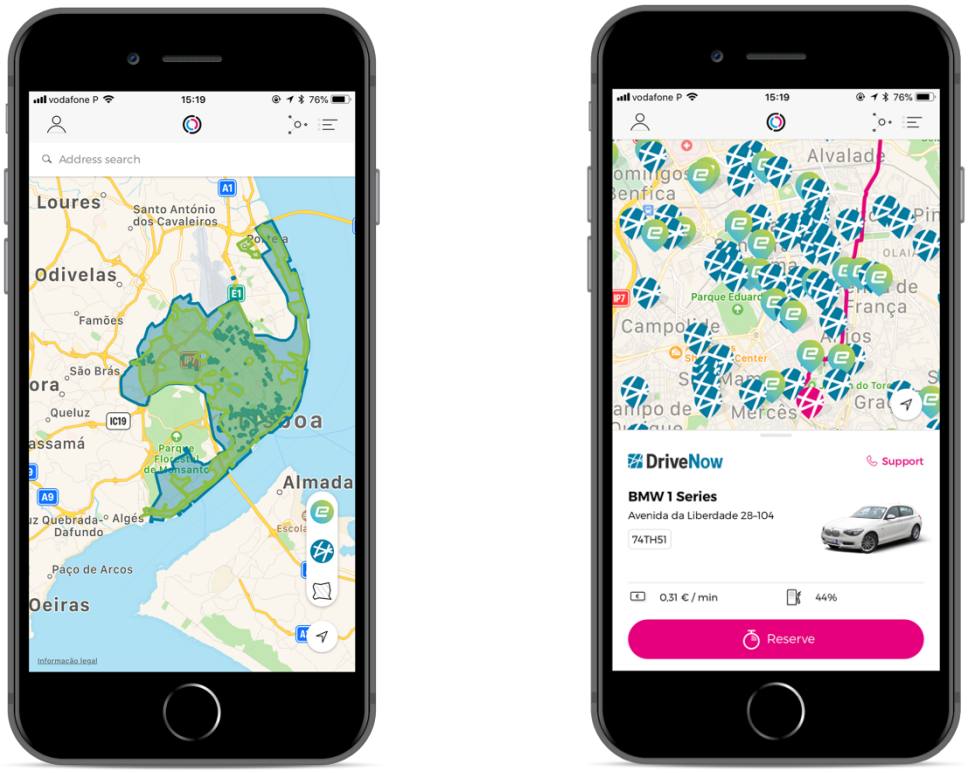


Figura 5 – Exemplo de Utilização do Free2Move
À esquerda: Área de atuação na zona de Lisboa
À direita: Oferta disponível na zona selecionada

2.5.3. BellHop

A Bellhop[29], apresenta-se como uma aplicação móvel gratuita, onde estão incluídos vários provedores de serviços de *ridesharing*. Estando disponível para descarga em vários países, a aplicação conta com várias *features* úteis quando se procura uma viagem.

Os principais diferenciais, face às aplicações já apresentadas, consistem na possibilidade de definir o número de passageiros a ingressar na viagem, o tipo de veículo que se pretende (desde o mais citadino ao mais executivo), a possibilidade de ordenar os resultados obtidos por preço ou rapidez, bem como uma notável e intuitiva interface de utilizador. Além destas, a aplicação também apresenta a estimativa do preço da viagem, a distância da viagem e o número de minutos que o motorista demorará até se apresentar ao cliente. [86]

[29] https://bellhop.app

O único aspeto negativo a apontar é relativamente à adoção do Inglês como o único idioma disponível para navegar na aplicação. Uma que vez que um dos focos de utilização deste tipo de aplicações é o turismo, a aplicação devia adotar o idioma do sistema operativo, ou a possibilidade de um menu para trocar a mesma. Incoerências linguísticas também foram detetadas, como apresentado na Figura 6 onde, ao invés de *Grand Central Terminal*, foi apresentado "Estação Grand Central".

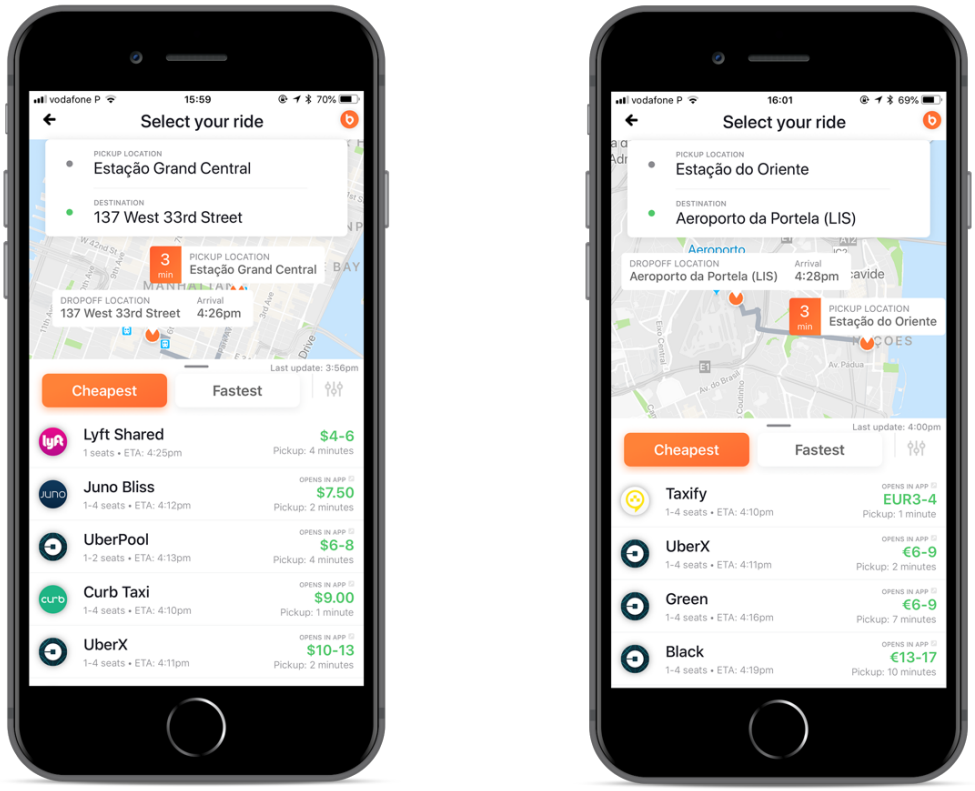


Figura 6 – Exemplos de Utilização do BellHop
À esquerda: New York, EUA
À direita: Lisboa, Portugal

É de notar que, o número de aplicações que efetuam a comparação de vários *providers*, não se limita a três. No levantamento das soluções existentes, é possível também referir a RideGuru[30], que compara serviços a nível mundial, contudo, com uma interface menos *user-friendly*, comparativamente às anteriormente analisadas, o que pode levar a uma comparação de serviços menos ágil.

[30] https://ride.guru

Também seria particularmente interessante testar as aplicações GoA2B[31] e Whipster[32], porém, estas não estão disponíveis na loja portuguesa de aplicações móveis, pelo que não foi possível o estudo das mesmas.

Apresentado assim este ecossistema subjacente à temática do *carsharing* e do *ridesharing*, a solução resultante do desenvolvimento desta dissertação tem como objetivo apresentar um sistema agregador, onde estejam incluídos o maior número de provedores de serviço, para que a oferta a apresentar a utilizadores futuros seja vasta, completa e sobretudo útil de modo a melhorar a sua experiência e adoção neste tipo de serviços.

Foram tomados em consideração não só provedores que operem nas cidades de Portugal, mas também de cidades de outros países, para que seja possível a posterior integração desta solução num maior número de projetos.

A integração da recolha de viagens provenientes de redes sociais, como é o caso do Facebook é outra valência presente no projeto que diferencia esta solução face às apresentadas no tópico anterior, 2.5 – Análise de plataformas agregadoras de mobilidade. A adoção do Facebook como ferramenta de partilha de viagens tem apresentado um crescimento favorável, não são em Portugal, como em outros países, pelo que o desenvolvimento do modelo responsável da extração das publicações que ofereçam a partilha de viagens seja um serviço ainda não explorado em outras soluções, tornadas públicas, e que está presente na solução desenvolvida nesta dissertação.

Assim, e uma vez que é espectável que o resultado das consultas aos *providers*, integrantes na solução, sejam diferentes entre si, a criação de um modelo genérico é uma das etapas vitais para a construção do modelo proposto. Este modelo reunirá assim as informações que se apresentam idênticas entre os diferentes providers, de modo a que o resultado final consiga oferecer um equilíbrio entre os dados recolhidos.

[31] http://www.goa2b.co

[32] http://www.whipstermobile.com

CAPÍTULO **3**

Levantamento de Requisitos

O presente capítulo é dedicado, em grande parte, à análise dos requisitos funcionais e não-funcionais da API. Para auxiliar na compreensão dos requisitos, antecedentemente são apresentados os conceitos do domínio do problema.

Como tem vindo a ser apresentado até ao presente ponto, a finalidade da solução final desta dissertação, consiste no desenvolvimento de um agregador de soluções de mobilidade, podendo estas ser tanto de *carsharing*, como de *ridesharing*, oferecendo assim, uma alternativa viável à suspensão da utilização do veículo privado.

Com a realização deste sistema, projeta-se assim uma solução onde seja possível, de uma forma centralizada, realizar facilmente comparações às viagens oferecidas pelos provedores disponíveis na solução, para um dado destino. Com isto, evita-se que o utilizador tenha de conferir individualmente cada serviço, de modo a escolher o mais favorável para a sua deslocação, quer relativamente ao preço da mesma, quer no que diz respeito ao tempo de deslocação que o condutor demora até se apresentar junto dos passageiros, entre outros.

Desta forma, e para que seja então possível a consulta dos dados dos diferentes tipos de *providers*, é necessária uma conexão prévia aos seus serviços, de modo a adquirir as informações e estatísticas relativas a uma dada viagem. Estas conexões realizam-se, maioritariamente, através de pedidos HTTP às APIs dos próprios, ou através da utilização de métodos do SDK, desenvolvidos pelos mesmos. Contudo, nem todos os fornecedores deste tipo de serviços disponibilizam, de forma pública e gratuita, o acesso à consulta dos dados relativos às viagens que providenciam.

Parte destes, não têm arquitetada uma solução que realize a consulta de detalhes e estatísticas a desenvolvedores externos, e outros simplesmente não autorizam o acesso à API ou aplicam um custo para a obtenção de um *access-token*. Apesar disto, seria possível a obtenção destes mesmos dados através de um *crawler* que processasse o conteúdo HTML das respetivas companhias, porém estariam a ser violadas normas de privacidade, uma vez que, estavam a ser recolhidos dados de modo não autorizado, quando estes apenas se apresentavam para consulta.

A obtenção de ofertas através de redes sociais é outro desafio presente no desígnio da dissertação. Cada vez é mais recorrente a partilha de viagens através de grupos dedicados a esse fim, sobretudo na rede social Facebook. Foi a partir deste mote, que foi necessário desenvolver um interpretador de linguagem natural, de modo a capturar corretamente as informações publicadas pelo anunciante, como origem, destino, data, hora e preço.

3.1. Requisitos

A idealização da solução, ou seja, da API desenvolvida, é um dos passos indispensáveis para a concretização do objetivo lançado. Desses é possível apontar o sistema de base de dados que reiunará os dados recolhidos dos provedores presentes, o tipo de resposta a devolver a cada pedido, os mecanismos de segurança implementados de modo a filtrar a consulta de utilizadores não registados e a interface de gestão do administrador da solução, detalhados no Capítulo 4 - Desenho da API.

Desta forma, é possível dividir os tipos de requisitos em dois, os funcionais e os não funcionais, explanados de seguida.

3.1.1. Requisitos Funcionais

Os requisitos funcionais são requisitos que descrevem a funcionalidade, isto é, as funções que o sistema deve realizar, ou os serviços que se espera que o sistema desempenhe.

É possível assim apontar os seguintes:

1. **Recolha e apresentação das informações relativas às viagens disponíveis de *providers* presentes na solução**

Deve ser possível apresentar numa só resposta JSON, uma coleção de informações relativa às viagens disponíveis para o destino pretendido, de todos os *providers* presentes na solução.

2. **Recolha e apresentação das informações relativas às viagens disponíveis de um dado *provider***

Deve ser possível apresentar numa só resposta JSON, todas as informações relativas às viagens disponíveis para um dado destino, de um *provider* específico.

3. **Registo na base de dados das viagens recolhidas nos pontos 1 e 2**

Deve ser possível registar na base de dados todas as viagens recolhidas, antes destas serem retornadas ao utilizador.

4. **Alteração de informações de uma viagem relativa a um dado *provider***

Possibilidade de alterar um ou mais campos de informações relativas a uma dada viagem de um dado provider, através do seu UUID, e registar novamente na base de dados.

5. **Criação uma nova viagem para um dado provider e registá-la na base de dados**

Possibilidade de criar uma viagem para um determinado *provider*, especificando todos os campos necessários e registar essa nova entrada na base de dados.

6. **Eliminar uma viagem através do seu UUID**

Possibilidade de eliminar uma viagem, previamente registada na base de dados, através da introdução do UUID.

7. Registo e obtenção um *access-token*

Um utilizador deve ter a possibilidade de se registar na plataforma e de obter um *access-token* para realizar os seus pedidos.

8. Interface de administrador

A solução deve incluir uma interface reservada para o administrador da mesma, onde será possível a gestão da infraestrutura da solução.

3.1.2. Requisitos Não-Funcionais

Os requisitos não-funcionais são requisitos que não correspondem diretamente à funcionalidade do sistema, mas expressam propriedades ou restrições sobre os serviços ou funções por ele providas. Alguns exemplos possíveis de apontar são:

1. Suporte a vários utilizadores

O sistema deve suportar utilizadores registados para que estes obtenham um *access-token*, de modo a realizar as suas consultas.

2. Arquitetura Orientada a Serviços

Todo o software deve seguir uma arquitetura orientada a serviços, com base em serviços da Web, para que o sistema possa ser facilmente estendido a várias plataformas, como um cliente, como interfaces nativas de dispositivos móveis ou soluções *in-browser*. Isto permite assim que o sistema se integre facilmente com outras plataformas.

3. Arquitetura de micro-núcleo

O sistema central deve ser conciso e estável, com suporte à adição de extensões e bibliotecas que permitem a adição de funcionalidades e melhoram o sistema.

4. Fácil integração em outras soluções

A solução deve ser de fácil integração, independentemente do número de outras soluções e do ambiente, mantendo assim a portabilidade em apenas um código base.

5. **Todas as consultas devem acompanhadas de um *access-token* para garantir que o acesso seja restrito a utilizadores que estejam registados na solução**

Todas as consultas (GET, POST, PUT, DELETE) realizadas à API devem ser acompanhadas de um *access-token*, introduzido no cabeçalho do pedido, que valida o registo prévio do utilizador na solução. Com isto é garantido a confidencialidade às APIs dos *providers* acedidas.

6. **Mensagens de erro auto-descritivas**

A solução deve, em caso de erro ou indisponibilidade de algum serviço, fornecer na resposta devolvida ao utilizador uma mensagem descrevendo o sucedido e o código de estado HTTP alusivo ao mesmo.

7. **Controlo de Versões**

O controlo de versões tem a finalidade de gerir diferentes versões de um projeto. Através deste é possível, de um modo inteligente, organizar o mesmo, uma vez que é possível acompanhar o histórico de desenvolvimento da solução e a colaboração concorrente, onde possibilita aos desenvolvedores trabalharem paralelamente sobre os mesmos ficheiros, apresentando as alterações efetuadas a cada versão.

3.2. User Stories

Os *user stories* (US) têm a finalidade de documentar o propósito da solução. Por outras palavras, descrevem as principais funcionalidades do sistema e a interação dessas funcionalidades com os diferentes atores, isto é, os tipos de utilizadores do sistema.

Uma vez que a solução final dispõe de dois atores – administrador e cliente – são explanadas as ações possíveis de desempenhar de cada ator, e as mesmo são apresentados de seguida. De notar que, as ações e privilégios que os diferentes atores possuem são notoriamente destintas.

Enquanto que o cliente apenas tem acesso à consulta de viagens, posteriormente à obtenção do *access-token* que permite a realização das mesmas, o administrador apresenta um número de privilégios maior. Enquanto administrador e no que diz respeito às viagens, este tem o potencial de adicionar, alterar e eliminar uma dada viagem, tanto através da interface de administrador, como pelos métodos da API alusivos a essas ações.

As restantes atividades traduzem uma prática relativa à gestão da infraestrutura, como:

- Gestão de clientes e os seus privilégios;
- Gestão dos *access-tokens* atribuídos a cada cliente;
- Gestão dos identificadores dos grupos de Facebook, para posterior realização dos *crawlers;*
- Gestão das chaves que permitem a conexão às APIs dos *providers* presentes na solução.

US 1. Enquanto utilizador, deve ser possível o registo na solução de modo a receber um *access-token* que permitirá a conexão à API.

US 2. Enquanto utilizador, deve ser possível o *login* na solução, de modo a renovar o *access-token* previamente gerado.

US 3. Enquanto utilizador, deve ser possível a conexão à API e a obtenção de informações relativas às viagens disponíveis de um determinado *provider*.

US 4. Enquanto utilizador, deve ser possível a conexão à API e a obtenção de informações relativas às viagens disponíveis por parte de todos os *providers* presentes na solução.

US 5. Enquanto utilizador, deve haver a possibilidade de ordenar o resultado da consulta, através do preço das viagens, tanto de modo ascendente como descendente.

US 6. Enquanto utilizador, deve haver a possibilidade de indicar qual o provider que aparecerá em primeiro, no resultado da consulta.

US 7. Enquanto utilizador, deve haver a possibilidade de indicar o número de lugares pretendidos para uma determinada viagem, filtrando assim, viagens desnecessárias.

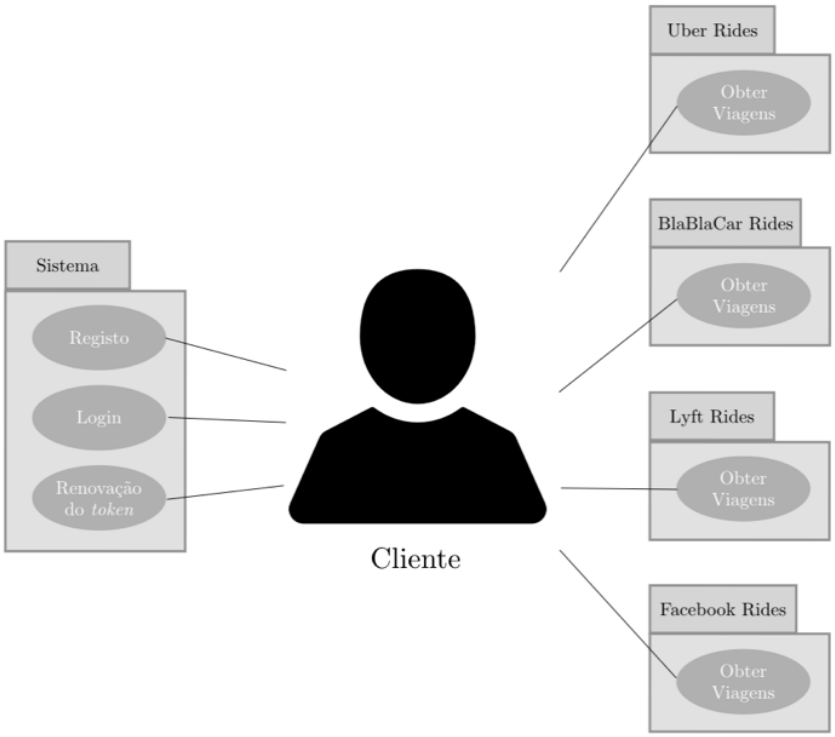


Figura 7 – Diagrama de Use Stories (Cliente)

US 8. Enquanto administrador, deve ser admissível o acesso a uma área reservada, onde seja possível gerir e aplicar as ações de CRUD a todas as já viagens armazenadas, aos registos relativos aos utilizadores e aos *access-tokens* atribuídos a estes, bem como aos *tokens* que permitem a conexão aos *providers* e às redes sociais a realizar o *crawler*.

US 9. Enquanto administrador, deve ser possível a conexão à API e realizar as ações de CRUD, ou seja, obter, adicionar, editar e remover uma dada viagem de um determinado *provider*.

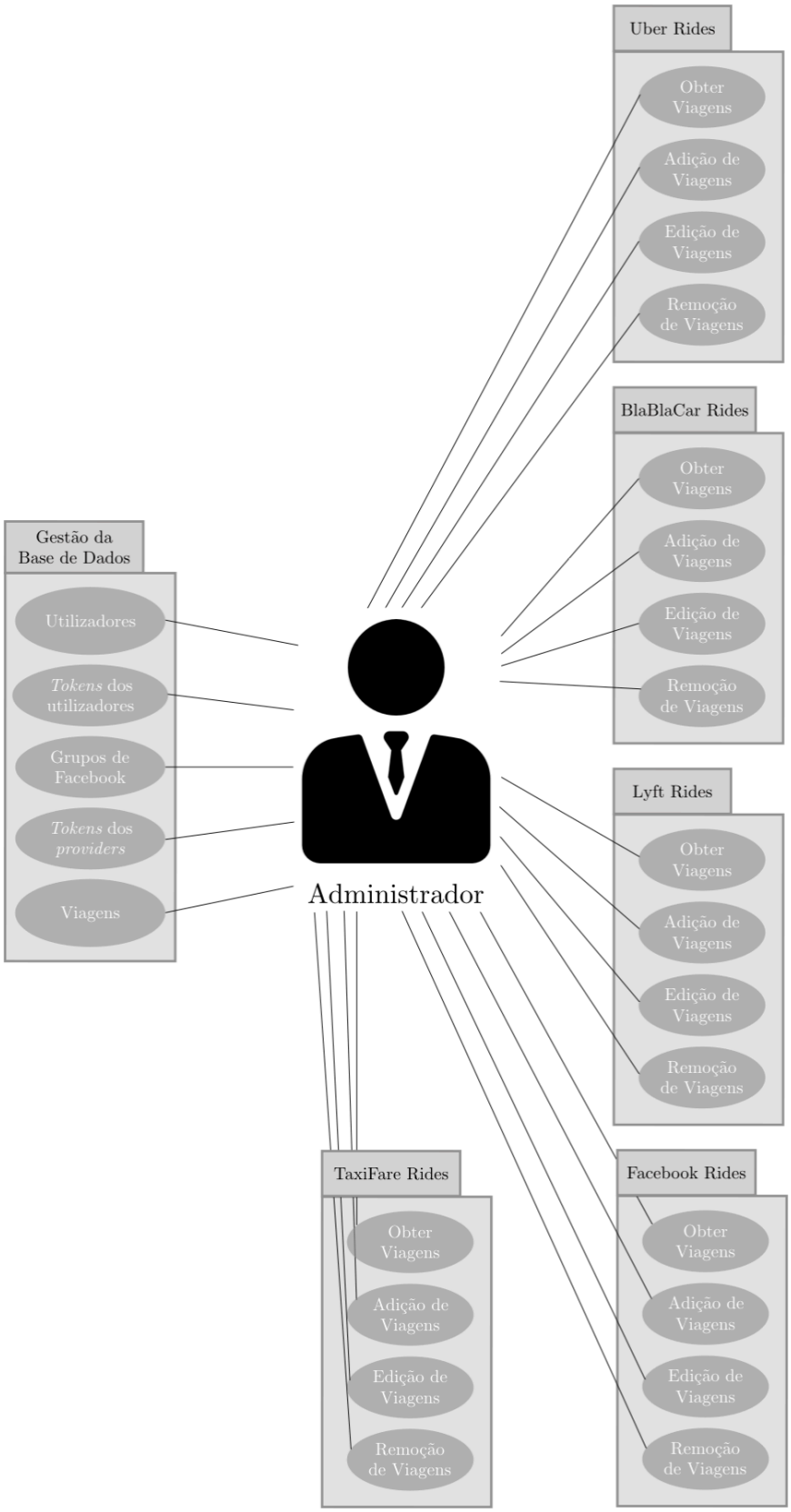


Figura 8 - Diagrama de User Stories (Administrador)

3.3. Análise de Redes Sociais

Como já tem vindo a ser apresentado, parte da missão desta dissertação centra-se na recolha de viagens que tenham sido publicadas em redes sociais. Assim, e de modo a verificar a afluência da partilha de viagens nestas plataformas, foi realizado um estudo nas redes sociais mais utilizadas de momento – Facebook[33], Instagram[34] e Twitter[35] – de modo a perceber a adoção desta prática pelos utilizadores destas.

Terminado esse estudo, foi bastante notório que a rede social que oferece um maior suporte para a partilha de viagens, é o Facebook.

Através da partilha em grupos e em páginas destinadas a esse fim, e onde um utilizador pode estar agregado, a procura destas torna-se num processo mais rápido e assertivo. Por sua vez, tanto os grupos como as páginas existentes, normalmente segmentam as cidades onde as viagens se realizarão, como por exemplo: "Boleias Lisboa-Aveiro e Aveiro-Lisboa", facilitando ainda mais a procura de uma boleia para um determinado destino.

O processo de divulgação é, habitualmente, expresso através de publicações e um exemplo de oferta de uma viagem, é apresentado na figura seguinte.

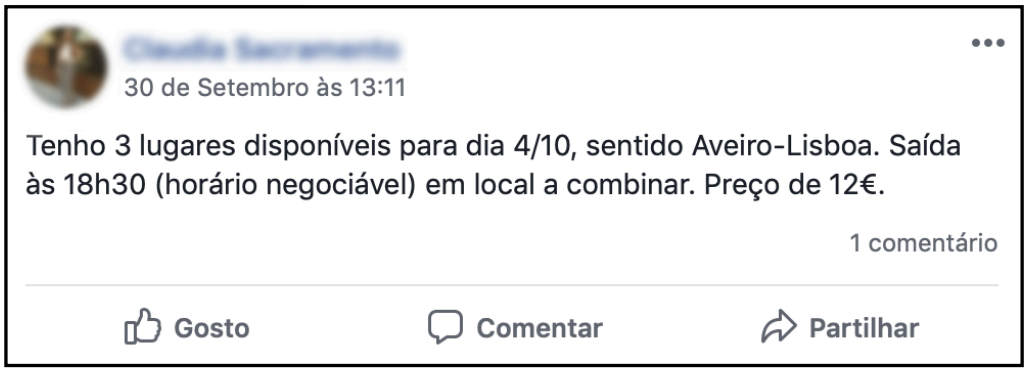


Figura 9 – Exemplo da oferta de uma boleia na rede social Facebook

[33] https://www.facebook.com

[34] https://www.instagram.com

[35] https://www.twitter.com

O processo de reserva ocorre tanto pelo contacto direto entre o condutor e o passageiro, pelo sistema de troca de mensagens Facebook Messenger[36], como através do sistema de comentários associados a cada publicação, permitindo assim a estes dois atores, por exemplo, agendar da melhor forma o local de partida.

Referentemente à análise da rede social Instagram, e sendo esta uma plataforma dedicada à partilha de fotografias e vídeos, a mesma foi selecionada para estudo, uma vez que utiliza o sistema de *hashtags*[37]. Através deste recurso, seria possível a procura da palavra-chave boleia e, deste modo, extrair o conteúdo presente na descrição associado a uma dada fotografia. Contudo, da pesquisa realizada, não foi encontrado nenhum registo semelhante ao apresentado anteriormente, onde existia a oferta de uma viagem e indicados os detalhes da mesma, pelo que a integração da rede social Instagram, foi descartada.

Isto não invalida, por exemplo, que um utilizador da plataforma não publique a oferta de boleias para os seus seguidores. A partilha tanto pode acontecer através da publicação de uma fotografia no meu mural, indicando na descrição os detalhes da mesma, ou através das *Stories*, uma ferramenta da plataforma, onde são publicadas imagens com uma duração de até 24 horas. Porém, entenda-se que estas, apenas estão acessíveis ao número de seguidores da pessoa que realizou a partilha.

Por último, a plataforma Twitter, consiste numa rede social onde são partilhadas mensagens até 280 caracteres e onde é também possível a adoção de *hashtags*. Com a mesma filosofia de procurar boleias através do uso de *hashtags*, como intencionado na rede social Instagram, não foram encontradas também correspondências que expressassem a oferta de viagens. Desta forma, e do mesmo modo que o Instagram, a integração do Twitter não será incluída na solução final.

[36] https://www.messenger.com

[37] Palavras-chaves ou termos de elevado interesse onde um utilizador consegue realizar pesquisas através da introdução das mesmas

3.3.1. Estratégias de extração de informações em Redes Sociais

Terminado assim o processo de análise das redes sociais detalhado anteriormente, o Facebook será a única plataforma que integrará a solução, de modo a ser possível a extração de dados relativos à partilha de viagens.

Desta forma, e de modo a proceder a esta extração, são normalmente utilizados *crawlers*, que podem ser definidos como um software, ou biblioteca de recolha de dados presentes em páginas *web*, permitindo assim a recolha de texto, imagens, vídeos, entre outros [87].

Atualmente, a biblioteca que permite a extração dos dados contidos na rede social Facebook, é uma ferramenta desenvolvida pelos próprios, denominada Graph API[38].

Contudo, é importante mencionar que até Maio de 2018, a conexão à Graph API esteve acessível e disponível para realizar consultas e recolher dados de perfis, grupos e páginas, cujos conteúdos estivessem assinalados como públicos, permitindo desta forma a extração de informações relativas a partilha de viagens. No entanto, e após tornado público o escândalo da recolha de dados de 87 milhões de utilizadores da plataforma, por parte da Cambridge Analytica, que estariam alegadamente a ser utilizados de modo a influenciar os utilizadores relativamente ao processo eleitoral dos Estados Unidos da América de 2016, o Facebook limitou drasticamente a plataforma de desenvolvedores, facultando acesso apenas a empresas e a aplicações certificadas [88] [89].

[38] https://developers.facebook.com/docs/graph-api

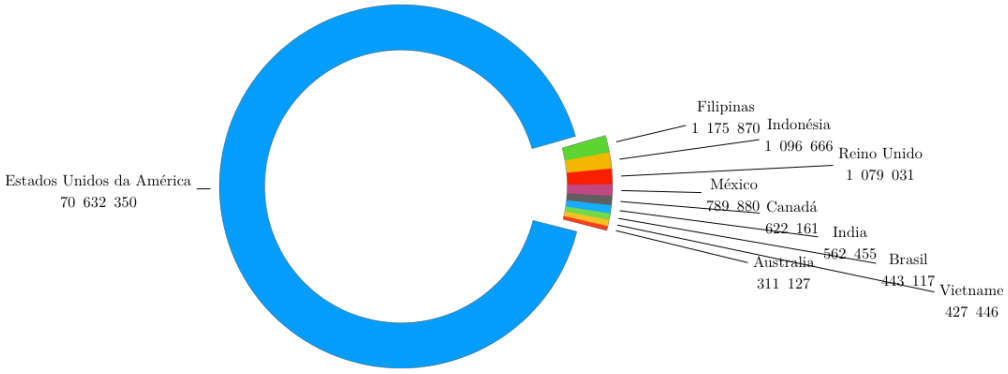


Figura 10 – Distribuição do número de afetados do ataque da Cambridge Analytica por país adaptado: https://goo.gl/QnYLLY

É partir da Graph API, com o devido acesso facultado, que será possível a recolha das informações que expressam o propósito de oferta de uma viagem, como o nome do utilizador que realizou a publicação, o corpo da publicação, bem como a data associada à mesma.

Contudo, é de notar que, apenas a extração do corpo da publicação, não se torna suficiente para identificar individualmente os elementos origem, destino, data, hora e preço de uma viagem. Para que esse fim seja atingido, é necessário aplicar técnicas de processamento de linguagem natural a cada *post* recolhido, como a tokenização e o uso de expressão regulares, de modo a reter individualmente cada um destes. Todos os detalhes associados a este processo estão descritos no tópico 4.4 – Análise e Processamento de Linguagem Natural em Redes Sociais.

De modo a evitar que este processo seja efetuado aquando da realização de uma consulta à API, teve que ser idealizado um método, que reduzisse a latência na obtenção dos dados e na entrega da resposta ao cliente. O processo de recolha e de tratamento dos dados recolhidos pode ser bastante lento e penoso, quando existem vários grupos para processar, podendo atrasar em largos minutos a resposta a apresentar ao cliente.

Assim, e de modo a contornar esta situação, é implementado na solução, um mecanismo responsável por executar todas as etapas anteriormente detalhadas, relativas à recolha e tratamento das publicações existentes nos grupos do Facebook, de uma em uma hora. Desta forma, o *crawler* trabalha em comunhão

com uma *framework* que permita a realização de tarefas periódicas, e será executado em *background* desde a iniciação do projeto no intervalo definido, armazenando na base de dados os resultados do mesmo, evitando que a cada consulta à API, todo este processo seja realizado. Neste caso, sempre que for realizado um novo pedido, serão devolvidas as entradas registadas na base de dados relativas à origem e ao destino pretendidos.

3.4. Revisão das Tecnologias

Para a concretização dos desafios propostos foram estudadas as principais tecnologias a integrar na solução, e os seguintes pontos apresentam as mesmas.

3.4.1. API

Como já apresentado, o produto final resultante da elaboração desta dissertação é uma API. À vista disso, o termo API refere-se assim a uma ferramenta, ou biblioteca, que permite o auxílio no desenvolvimento de código que interaja com outras soluções já existentes.

Ao longo dos últimos anos, as APIs têm sido alvo de grandes avanços. Temáticas como a internet das coisas, aplicações móveis, redes sociais ou *cloud computing*, são exemplos de tecnologias que recorrem amplamente a *Web* APIs. A relevância que as APIs estão a ter atualmente, também se pode verificar através do crescimento exponencial de *Web* APIs públicas [90], que são facilmente integráveis em aplicações terceiras. Na figura seguinte, é possível constatar o crescimento que as *Web* APIs públicas tiveram desde o ano 2005, com apenas um registo, até ao fecho do ano 2017, registando, segundo a ProgrammableWeb[39], 18595 APIs. Atualmente, a plataforma já conta com um histórico superior a 20000 APIs, onde a média de registos mensais é de 172, atingindo 2060 num ano [91].

Apesar do enorme crescimento registado, estes números são escassos, no que incumbe às APIs existentes, uma vez que não estão contabilizadas as APIs privadas.

[39] https://www.programmableweb.com

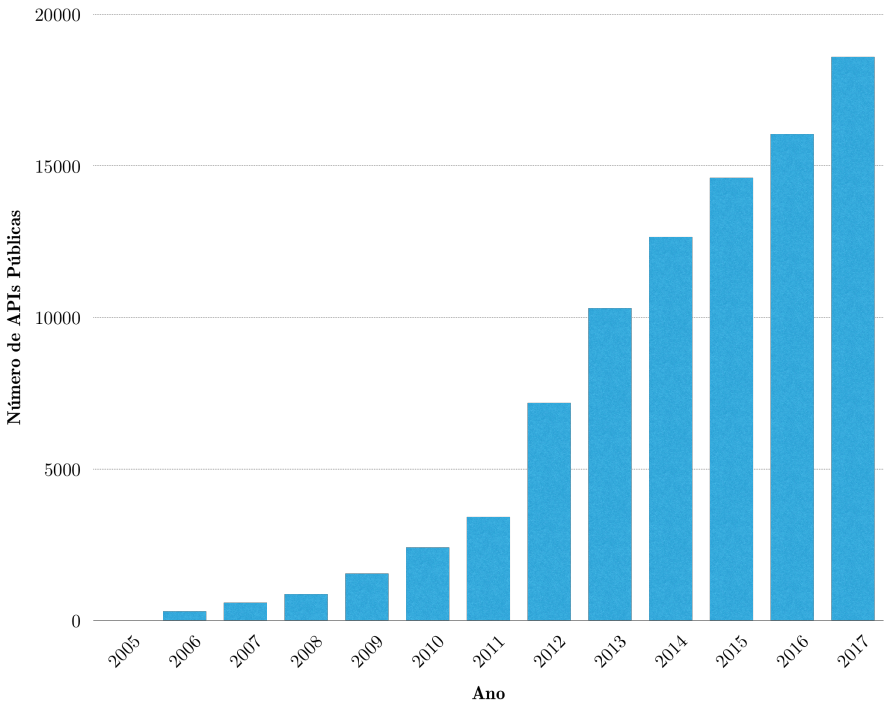


Figura 11 – Crescimento do número de APIs públicas ao longo dos anos adaptado: https://goo.gl/Y9Yi4N e https://goo.gl/yrUuYm

O uso deste tipo de soluções, no mundo empresarial, deu-se sobretudo quando as organizações começaram a sentir a necessidade de integrarem os seus sistemas internos. Este caminho, trilhado pelas mesmas, contribuiu fortemente para o crescimento do número de APIs, que se verifica até aos dias de hoje.

Naturalmente, o tipo ou estilo das APIs desenvolvidas, foram também evoluindo ao longo dos anos. O conceito por detrás das APIs, já existe desde os primórdios da computação, porém só a partir dos anos 2000, é que começaram a surgir as primeiras Web APIs. Nos últimos anos, o estilo predominante para o desenvolvimento de APIs, e também o adotado no desenvolvimento deste projeto, é o *Representational State Transfer* – REST –, como observável na Figura 12.

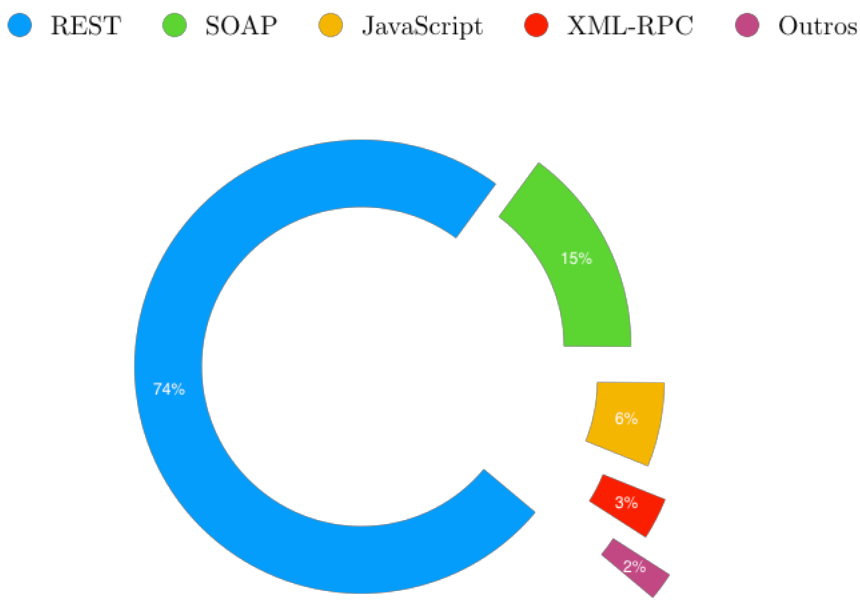


Figura 12 – Estilos arquitetónicos do desenvolvimento de Web APIs
Estudo conduzido em Maio de 2011
adaptado: https://goo.gl/KhFKnJ

3.4.2. Arquitetura REST

As APIs baseadas nos estilos apresentados na Figura 12, são exemplos de uma arquitetura orientada a serviços – *Service-Oriented Architecture* (SOA). Este é um padrão para o desenvolvimento de sistemas distribuídos que, genericamente, disponibiliza serviços para outras aplicações ou serviços, por meio de um protocolo de comunicação previamente estabelecido, sendo normalmente HTTP ou HTTPS. A lógica por trás dessas arquiteturas, centra-se no facto das aplicações estarem conectadas a serviços e não a outras aplicações. Partindo deste facto, em que todos os componentes podem ser vistos como um serviço independente, o desenvolvimento da solução final do Agregador de Serviços de Mobilidade Alternativa, baseou-se neste paradigma.

Assim, e com uma arquitetura orientada a serviços, é possível obter vários benefícios, de modo a auxiliar as organizações a manterem-se competitivas. A tabela seguinte sumariza os principiais benefícios subjacentes a uma arquitetura orientada a serviços [92].

Tabela 4 – Benefícios da Arquitetura Orientada a Serviços

Benefício	**Descrição**
Fácil integração e gestão da complexidade	Tendo o seu foco no serviço e não na implementação, uma SOA fornece transparência na implementação e minimiza o impacto quando ocorrem alterações na infraestrutura e na implementação.
Comercialização do produto mais rápida	A capacidade de compor novos serviços a partir dos já existentes proporciona uma vantagem distinta para uma organização dar resposta a necessidades comerciais exigentes. Com isto, conquista-se uma redução no tempo necessário no levantamento de requisitos, desenvolvimento do software e em testes. Isto leva assim ao rápido desenvolvimento de novos serviços de negócios e permite que uma organização responda rapidamente a mudanças e reduza o tempo de comercialização do produto, e futuros custos de manutenção.
Foco no Futuro	Os processos de negócios que fazem parte de uma série de serviços de negócios podem ser criados, alterados e geridos com mais facilidade para atender às necessidades que possam surgir. Uma SOA fornece a flexibilidade e capacidade de resposta que é fundamental para as empresas sobreviverem e prosperarem.

Porém, é também possível apontar desvantagens relativamente às arquiteturas dos SOA, e a mais significativa é a fragilidade das alterações nos protocolos de comunicação. Mudanças na API, podem comprometer totalmente a solução entregue ao cliente e, portanto, as APIs devem ser cuidadosamente desenvolvidas e tornadas extensíveis, de modo a não quebrar a compatibilidade. [92] [93]

Comparação entre os estilos SOAP e REST

Ao longo dos anos, e com a evolução da tecnologia, vários têm sido os padrões explorados de modo a promover o desenvolvimento de arquiteturas orientadas a serviços. Destes, é possível apresentar dois que permitem a comunicação entre o *frontend* e o *backend*, o SOAP – *Simple Object Access Protocol* – e o REST.

O **SOAP** consiste num protocolo aplicado a ambientes distribuídos e descentralizados, isto é, permite que dois processos possam comunicar estando a ser executados em máquinas ou redes diferentes [94]. Este é um protocolo de transferência de mensagens, no formato XML, que permite a comunicação entre os diferentes componentes, sendo independente da plataforma e do protocolo de transporte, como é o caso do HTTP, TCP e SMTP [95].

Por sua vez, o **REST** é considerado uma alternativa menos formal que a anteriormente apresentada, onde requer apenas o uso do protocolo HTTP, assumindo uma comunicação direta de ponto a ponto. Apresenta também como principais vantagens uma maior facilidade de implementação, escalabilidade e *performance* na sua execução, e sem a complexidade dos *Web-Services* tradicionais [96].

Relativamente aos tipos de formatos, o REST poderá utilizar diferentes tipos, sendo que, o formato das mensagens deve ser definido de acordo com a linguagem/aplicação que está a ser desenvolvida; enquanto o SOAP está limitado ao XML, como formato de resposta. Apesar do SOAP ser uma arquitetura mais fácil de ser interpretada, uma vez que os dados estão armazenados em elementos XML, o documento XML torna-se mais pesado devido a uma maior quantidade de informação a ser transportado entre os sistemas, exigindo assim uma maior capacidade de processamento, o que consequentemente, exigirá mais tempo. Em contrapartida, o REST não utiliza elementos XML para identificar os recursos, ocupando assim menos espaço. Os recursos e as diferentes operações sobre os mesmos estarão disponíveis através de URLs, onde a transferência e o processamento dos dados são mais rápidos [95].

Uma vez que é considerado um dos tipos de *Web-Service* mais rápido, objetivo, simples e de fácil aprendizagem e utilização, o REST ganhou uma maior popularidade no desenvolvimento e na disponibilização de APIs de acesso público, conforme ilustrado na Figura 12.

Desta forma, e em jeito de sumarização, apesar das vantagens e desvantagens que cada tipo de *Web-Service* apresenta, é possível concluir que o REST é a melhor opção para os casos onde existe a limitação de recursos e de largura de banda, para realizar operações CRUD sem estado e para casos onde a informação precisa ser guardada em *cache*. É um serviço extremamente flexível uma vez que não existe restrições no formato em que a informação é devolvida, mas sim no comportamento dos componentes envolvidos. O SOAP, por outro lado, é uma solução mais indicada para instituições com padrões a seguir rígidos e ambientes complexos onde a integridade, segurança e confiabilidade dos dados são fatores importantes a ter em conta na transferência dos mesmos [97].

Em termos de segurança, tanto a arquitetura REST como a arquitetura SOAP pode ser implementada utilizando o protocolo HTTP sobre *Secure Sockets Layer* (SSL) para proporcionar comunicações seguras. A arquitetura SOAP contém ainda um conjunto de opções que podem ser aplicadas à troca de mensagens entre os diferentes módulos de um sistema, permitindo assim, que o serviço seja dotado de um nível de qualidade e segurança na comunicação de uma forma independente do protocolo de transporte utilizado [98].

Arquitetura REST

O REST, como já apresentado, é um estilo de arquitetura desenvolvido para servir aplicações *Web*, oferecendo um conjunto de linhas orientadas necessárias para desenvolver um serviço coeso, escalável e com elevada *performance*, onde o resultado é independente da plataforma e da linguagem.

No ano de 2000, Roy Fielding, contribui consideravelmente com a apresentação da sua tese de doutoramento – *Architectural Styles and the Design of Network-based Software Architectures* [96] –, para o desenvolvimento de sistemas baseados na *Web*, onde generalizou os princípios da arquitetura da *Web* e apresentou-os como um estilo arquitetural.

Descreveu que a arquitetura REST é derivada de um conjunto de princípios bem conhecidos, como o cliente-servidor, *stateless*, *cache*, interface uniformizada e o sistema em camadas [99]. Ao seguir as mesmas, é suposto que o resultado seja um sistema altamente escalável, com capacidade para crescer organicamente de uma maneira descentralizada.

Através das representações com recurso a URLs únicos, é possível empregar as operações GET, POST, PUT e DELETE, disponíveis no protocolo HTTP, de modo a melhorar o acesso a dados remotos. Deste modo, os desenvolvedores têm a possibilidade de configurar a resposta em HTML, XML, JSON, entre outros. Enquanto isso, as páginas *web* são projetadas para um consumo frequente e rápido, o que significa que os *browsers* apenas implementam a operação GET para o carregamento de dados, enquanto que os serviços REST, utilizam todas as operações, de modo a impulsionar a comunicação máquina-máquina.

A aglomeração de serviços, pela qual o REST é caracterizado, permite a integração e interoperabilidade[40] pretendida. Neste tipo de arquitetura, o objetivo é responder, de forma natural, a todos os pedidos, permitindo uma melhoria contínua da solução, sem obstruir a solução existente. Disto resulta uma maior flexibilidade, escalabilidade e capacidade de resposta.

Para além das vantagens apresentadas, existem certas inconveniências no serviço. Uma dessas, é a possibilidade de ocorrência de problemas de conetividade. Sabendo que a conexão à internet, é a principal linha de comunicação, é necessário garanti-la 24 horas por dia.

Um exemplo possível de apresentar, é o desenvolvimento de aplicações móveis, que dependem de um serviço remoto para sua correta execução, onde a usabilidade da aplicação fica comprometida em situações em que a conexão à internet não esteja disponível. Outra eventual inconveniência a apontar, é a possível redução no desempenho e robustez, devido à paralelização que a execução do serviço toma, ou seja, o *workflow* de cada serviço, necessita aguardar pela conclusão do serviço anterior antes de prosseguir.

Em jeito de conclusão, o termo das arquiteturas orientadas a serviços, apresenta-se para um conceito *everything-as-a-service*. Este novo paradigma requer assim esforços, tanto para consumidores, como para desenvolvedores de serviços.

[40] capacidade de um sistema se comunicar de forma transparente com outro sistema

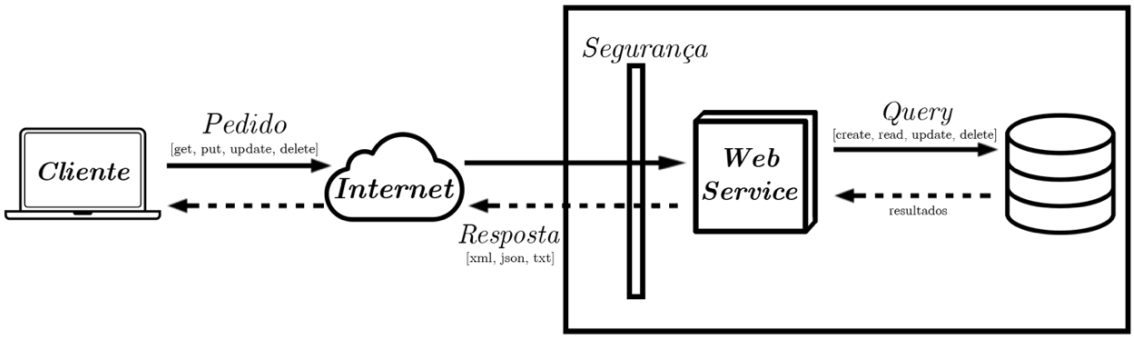


Figura 13 – Esquematização da Arquitetura REST
adaptado: https://goo.gl/oCjf5k

3.4.3. HTTP

Como expresso anteriormente, o protocolo de transporte utilizado pela arquitetura REST, é o HTTP – *Hypertext Transfer Protocol* – um protocolo inserido na camada aplicacional do modelo TCP/IP, para a transferência de informação entre sistemas distribuídos, geralmente entre *Web-Servers* e utilizadores. É caracterizado por ser um protocolo genérico e aplicável em diversos contextos, onde providencia uma interface uniforme para interagir com os recursos, através da manipulação e transferência de representações. Esta interface é independente dos recursos fornecidos, o que significa que não depende da natureza do recurso ou da sua implementação, permitindo assim esconder os detalhes de implementação de um serviço. [100]

O modelo de comunicação que este protocolo utiliza, é o modelo cliente-servidor, e a comunicação entre ambos é realizada em duas etapas. Primeiramente, o cliente efetua um pedido HTTP e o servidor, ao receber o mesmo, trata-o e envia uma resposta HTTP.

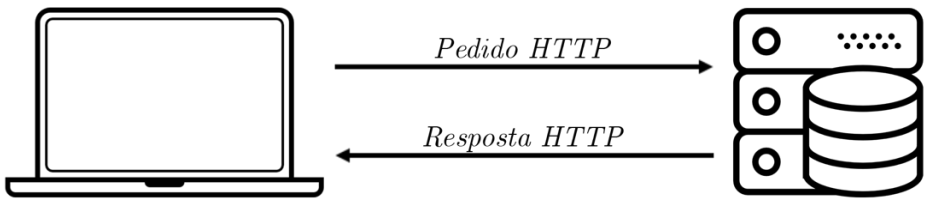


Figura 14 – Modelo Cliente-Servidor
adaptado: https://goo.gl/8eFPU5

Com o propósito de distribuir informação pela Internet, este tem sido usado desde 1990, na sua primeira versão – 0.9, onde era apenas possível solicitar dados em formato de texto ASCII, através do método GET.

O protocolo foi sofrendo evoluções ao longos dos anos, até chegar à presente versão, a 2.0, devido à necessidade de vencer a necessidade da transferência de dados apenas em texto, passando assim a suportar mensagens do tipo MIME44 – *Multipurpose Internet Mail Extension* – e de suporte a novos métodos de requisição, como o POST e HEAD. [101]

Em seguida, será explicada parte da semântica do protocolo HTTP, notando que esta semântica tem como base a versão 1.1 do protocolo, atualmente a mais utilizada e que se mantém na versão 2.0.

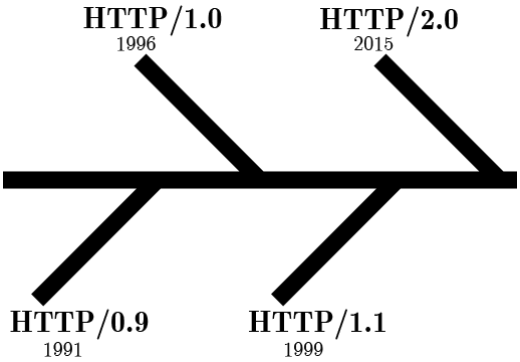


Figura 15 – Linha Temporal da Evolução dos Protocolos HTTP

Mensagens

As comunicações no protocolo HTTP, entre cliente e servidor, processam-se através de mensagens. Quando uma mensagem é enviada de um cliente para o servidor, é classificada como um pedido HTTP, e uma mensagem que é enviada de um servidor para um cliente, é classificada como uma resposta HTTP. Neste contexto, um cliente pode ser descrito como um programa que estabelece uma conexão a um servidor, com o propósito de enviar um ou mais pedidos HTTP; um servidor, é um programa que aceita conexões, de modo a receber os pedidos HTTP e a enviar respostas HTTP. Os termos cliente e servidor, referem-se apenas aos papéis que estes programas desempenham para uma conexão em particular, podendo acontecer que o mesmo programa aja, quer como cliente, em certas conexões, quer como servidor.

O HTTP, é usualmente referido como sendo um protocolo sem estado. Isto significa que cada transação é atómica, ou seja, não há nenhuma especificação que exija algum tipo de associação entre diferentes transações. Uma transação consiste assim, num único pedido HTTP e na resposta HTTP correspondente.

Métodos HTTP

Os métodos de solicitação, indicam o propósito dos pedidos realizados pelos clientes. O protocolo HTTP, define um total de oito métodos, que indicam ao servidor a ação a executar no recurso especificado.

A tabela seguinte – Tabela 5 – apresenta uma breve descrição associada a cada um destes.

Tabela 5 – Métodos HTTP

Método	Descrição
GET	Transfere a representação atual do recurso alvo especificado.
HEAD	Variação do método GET, onde apenas o cabeçalho é transferido, evitando a recuperação de todo o conteúdo. Pode também ser usado, para obter meta-dados relativos à representação selecionada, sem transferir os dados da mesma.
POST	Executa algum tipo de processamento, relativo a um dado recurso, no conteúdo do corpo do pedido e não na URI, como por exemplo, a submissão de um formulário *online*. Desta forma, oferece uma maior segurança em relação aos dados transferidos, ao contrário do método GET, onde os dados são anexados no URL.
PUT	Altera todas as representações atuais do recurso alvo pelo conteúdo do corpo do pedido.
PATCH	Aplica as modificações parciais, descritas no corpo do pedido, ao recurso alvo.
DELETE	Exclui todas as representações relativas a um dado recurso.
CONNECT	Estabelece um túnel SSL e TLS para o servidor identificado pelo recurso alvo, permitindo assim criar uma conexão segura.
OPTIONS	Retorna e descreve as capacidades suportadas pelo recurso alvo, ou seja, os métodos HTTP que o servidor aceita.
TRACE	Executa um teste, com o intuito de verificar como é que a mensagem é manipulada ao longo do caminho até ao recurso alvo.

adaptado: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content (Fielding R. e Reschke J., 2014)

Para além dos métodos contidos na especificação do HTTP/1.1, é também descrito o método PATCH, apresentado *à posteriori*, mas que assume uma importância no desenho e na implementação de uma API RESTful. Este método vem assim como resposta à necessidade de modificar, parcialmente, um recurso, dado que o método PUT é mais apropriado para a substituição integral de um recurso.

É também possível notar que, os métodos presentes na Tabela 6, partilham de algumas propriedades, entre as quais a segurança e a idempotência, bastante relevantes aquando o planeamento de uma REST API.

Relembrando que, um método é considerado seguro, se não produzir alterações no estado do servidor, quando um cliente faz um pedido. Um método seguro, não previne que o servidor se comporte de uma maneira inesperada, apenas determina que o cliente não pode ser responsabilizado por qualquer problema que ocorra no servidor, por este executar código potencialmente inseguro (por exemplo, apagar um registo da base de dados), em resposta a um método seguro.

Por outro lado, um método é idempotente se, após múltiplos pedidos idênticos, o efeito pretendido no servidor for o mesmo de um único pedido. A particularidade deste, é que o pedido pode ser repetido automaticamente caso ocorra uma falha de comunicação antes que o cliente consiga interpretar a resposta do servidor.

Desta forma, todos os métodos seguros também são considerados idempotente e, para além desses, o PUT e o DELETE.

Tabela 6 – Propriedades dos métodos HTTP

Método	**Segurança**	**Idempotência**
GET	√	√
HEAD	√	√
POST	×	×
PUT	×	√
PATCH	×	×
DELETE	×	√
CONNECT	×	×
OPTIONS	√	√
TRACE	√	√

adaptado: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content (Fielding R. e Reschke J., 2014)

Destes métodos apresentados, apenas o GET, HEAD, OPTIONS e TRACE são seguros, sendo o PUT e o DELETE idempotentes. Desta forma, quando é realizado um pedido PUT, o estado do servidor no final do pedido é igual ao estado verificado, caso o cliente faça vários pedidos PUT, sempre com os mesmos dados, pois os métodos PUT substituem todas as representações atuais do recurso alvo, pelas representações enviadas pelo cliente. [90] [102]

Códigos de Estado da Resposta HTTP

Ao responder a um pedido proveniente do cliente, o servidor inclui, na mensagem de resposta, um código composto por três dígitos. O propósito deste código é retribuir ao cliente o resultado da tentativa do servidor em perceber e satisfazer o pedido.

Existe um conjunto alargado de códigos que são categorizados de acordo com a resposta do servidor, tal como ilustra a tabela 7. Na tabela 8 estão referenciados alguns dos códigos de estado mais relevante no contexto de uma API REST.

O primeiro dígito do código de estado, define a classe da resposta, e os dois seguintes não possuem nenhum papel de categorização. Existem assim, cinco valores para o primeiro dígito. [90] [102]

Tabela 7 – Categorização dos códigos de estado de uma resposta HTTP

Classe	**Descrição**
1xx (Informativo)	O pedido foi recebido e está a ser processado.
2xx (Bem-Sucedido)	O pedido foi recebido com sucesso, compreendido e aceite.
3xx (Redirecionamento)	Informa que é necessária uma ou mais ações adicionais, para completar o pedido.
4xx (Erro do Cliente)	O pedido não está bem formulado sintaticamente ou não foi possível atender o pedido.
5xx (Erro do Servidor)	O servidor não conseguiu entender um pedido aparentemente válido.

adaptado: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content (Fielding R. e Reschke J., 2014)

Tabela 8 – Definição de alguns códigos de estado de uma resposta HTTP

Código	**Definição**
200 OK	O pedido foi executado com sucesso.
201 CREATED	O pedido foi executado com sucesso e, como resultado, foram criados um ou mais recursos que são identificados através do campo Location do cabeçalho da resposta, ou através do próprio URI do pedido.
400 BAD REQUEST	O servidor não pode ou não processará o pedido, devido a um problema presente no pedido realizado pelo cliente (Ex: sintaxe incorreta).
403 FORBIDEN	O pedido foi compreendido, porém, recusado.
404 NOT FOUND	Não foi encontrada uma representação para o recurso alvo, ou o servidor recusa-se a divulgar que existe.
405 METHOD NOT ALLOWED	O método especificado não é suportado para o recurso indicado pelo pedido.
408 REQUEST TIMEOUT	O servidor não recebeu nenhum pedido dentro do tempo que estava estipulado.
409 CONFLICT	O pedido não pode ser concluído, devido a um conflito com o estado atual do recurso alvo.
500 INTERNAL SERVER ERROR	O servidor encontrou um problema que o impede de prosseguir com o processamento do pedido.

adaptado: Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content (Fielding R. e Reschke J., 2014)

3.4.4. Frameworks REST

A utilização do estilo da arquitetura REST para o desenvolvimento de *Web-Services* tornou-se cada vez mais popular e, com isso, foram propostas e desenvolvidas diversas *frameworks* capazes de providenciar aos programadores uma forma simples e acessível de implementar soluções, seguindo uma arquitetura REST.

Para que fosse possível o desenvolvimento da REST API proposta, foi elaborado um estudo sobre as *frameworks* existentes, que permitissem tal ação.

Desta forma, foi tomado em consideração a dissertação desenvolvida por Filipe Perdigão de Souza – *Criação de framework REST/HATEOAS Open Source* para desenvolvimento de APIs em Node.js – [98], onde são apresentadas e detalhadas as principais *frameworks* existentes.

Na análise tomada, foram considerados fatores como a linguagem de programação em que é desenvolvida, dando maior preferência à linguagem Python, fatores como a autenticação, sistema de permissões e também a serialização dos dados, bem como, uma completa e intuitiva documentação disponível para os desenvolvedores.

No final desta análise, a *framework* que apresentou maiores valências de modo a permitir o desenvolvimento da REST API que cumprisse os fatores, previamente apresentados, é o Django Rest Framework (DRF)[41].

De seguida, são apresentadas duas tabelas comparativas, onde são apresentadas as *frameworks* e os fatores tomados em consideração no estudo.

Tabela 9 – Tabela comparativa de frameworks REST (Parte 1)

	Linguagem	**Baseada em Recursos**	**Serialização**	**Autenticação**	**Cache**
Django Rest Framework	Python	Sim	XML, JSON, HMTL	OAuth, OAuth2, Token, Sessão	Sim
Flask-RESTFul	Python	Sim	JSON	Não	Não
Restlet	Java	Sim	JSON, XML, CSV	Digest, Amazon S3, OAuth2	Não
Spark	Java	Não	Não	Não	Não
Sinatra	Ruby	Não	Não	Através de middleware Rack	Sim
Express	JavaScript	Não	JSON	Não	Não
Restify	JavaScript	Não	JSON	Através de assinatura	Sim
Sails	JavaScript	Sim	JSON	Através de extensões	Não
Loopback	JavaScript	Sim	JSON	Facebook, Google, GitHub, OAuth2	Não

adaptado: Criação de framework REST/HATEOAS Open Source para desenvolvimento de APIs em Node.js (Souza F., 2015)

[41] http://www.django-rest-framework.org

Tabela 10 – Tabela comparativa de frameworks REST (Parte 2)

	Permissões	**Filtros**	**HATEOAS**	**Logging**	**Documentação**
Django Rest Framework	Sim	Sim	Sim	Não	Intuitiva, Bastante Documentada
Flask-RESTFul	Não	Não	Não	Sim	Detalhada
Restlet	Sim	Sim	Não	Sim	Extensa, Detalhada
Spark	Não	Sim	Não	Não	Simples, Minimalista
Sinatra	Não	Sim	Não	Sim	Extensa, Simples
Express	Não	Sim	Não	Sim	Detalhada, Intuitiva
Restify	Sim	Sim	Não	Sim	Detalhada
Sails	Não	Sim	Não	Sim	Bastante Minimalista
Loopback	Sim	Sim	Não	Sim	Detalhada

adaptado: Criação de framework REST/HATEOAS Open Source para desenvolvimento de APIs em Node.js (Souza F., 2015)

Desta forma, para uma correta compreensão relativamente à framework Django Rest Framework, torna-se essencial uma apresentação e descrição prévia relativa à framework Django. As mesmas, são apresentadas nos tópicos abaixo.

3.4.5. Django

O Django[42], é uma *framework* de desenvolvimento web gratuita e *open-source*, escrita em Python, idealizada para promover um rápido desenvolvimento, preservando aspetos cruciais como a rapidez, segurança e a escalabilidade [103].

Segue uma arquitetura semelhante ao MVC (*Model-View-Controller*), mas aplicado a aplicações *web* – MVT (*Model-View-Templates*), que separa a representação da informação, da interação do utilizador com esta.

[42] https://www.djangoproject.com

Nesta, é possível [104]:

- descrever o modelo de dados, orientado a objetos, no *Models*. Cada classe gerada é equiparada a uma tabela de uma base de dados, onde são definidas as informações pretendidas, e onde as instâncias dessa mesma classe, representam os registos das tabelas. A *framework* Django facilita a recolha dos dados, permitindo ao desenvolvedor não se preocupar com a conexão entre as classes de domínio e a base de dados. Para isso, recorre à técnica designada por Mapeamento Objeto-Relacional, do inglês *Object-Relational Mapping* (ORM). O administrador da solução, conta ainda com uma intuitiva área reservada de consulta dos dados contidos na Base de Dados, podendo rapidamente aplicar as quatro operações essenciais de base de dados: CRUD – *Create*, *Read*, *Update* e *Delete*.

- controlar o que os utilizadores veem, através das *Views*. Neste ponto, são implementadas regras de apresentação e de negócio, da solução a desenvolver. Nestes métodos, é realizada a lógica associada e é retornada para um URL em específico, onde a informação retornada ao cliente pode ser tanto em XML, HTML, JSON, entre outros. É também aqui que é realizada a comunicação entre os *Models* e os *Templates*. Deste modo, as *views* são responsáveis por controlar o fluxo de informação entre os *models* e os *templates*, isto é, decide quais as informações a serem obtidas na base de dados, através do modelo e quais delas serão enviadas para a visualização.

- controlar como estes a veem, a partir dos *Templates*. Esta camada descreve a forma visual em que os dados são apresentados ao utilizador, apresentando os dados das camadas inferiores, sendo a única que permite a interação do utilizador. É maioritariamente composta por ficheiros HTML, CSS e JavaScript – linguagens focadas na representação gráfica para o utilizador. Comparando com o modelo MVC, esta camada corresponde ao *Controller*.

Outra particularidade interessante de referir é a valência para integrar bibliotecas já desenvolvidas, e integrá-las na solução através do maior e mais utilizado sistema de gestão de pacotes desenvolvidos em Python, o PIP[43].

[43] https://pip.pypa.io/en/stable

Com a utilização de bibliotecas já desenvolvidas pela comunidade, a solução torna-se mais rica e com um esforço menor, evitando assim a "reinvenção da roda". Com isto, é possível afirmar que a *framework*, no seu todo, assenta no princípio de DRY – *Don't Repeat Yourself* – onde faz com que o desenvolvedor aproveite ao máximo o código já desenvolvido.

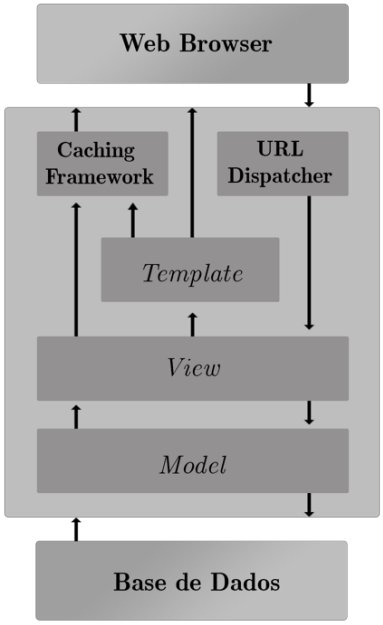


Figura 16 – Arquitetura do Django
adaptado: https://goo.gl/ca9xv2

Padrão ORM

O ORM é uma técnica que permite consultar e manipular os dados de uma base de dados, através de uma perspetiva orientada a objetos, adaptado à linguagem de programação que está a ser utilizada ao invés de recorrer a instruções SQL [105]. A figura seguinte, esquematiza a arquitetura associada a este processo apresentado.

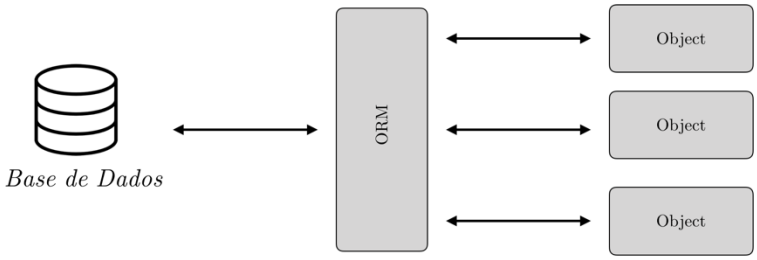


Figura 17 – Arquitetura do Padrão ORM
adaptado: https://goo.gl/DW44Cu

O ORM, mapeado nos *Models*, serve assim como uma ponte entre o modelo relacional, a base de dados, e as classes e métodos orientados a objetos, fazendo o mapeamento às tabelas que constituem a base de dados relacional para estruturas de dados. Convém notar que, o mapeamento é apenas realizado à base de dados e não aos registos. Os registos de cada tabela são representados por instâncias das classes correspondentes.

Existe assim uma independência dos modelos em relação à base de dados, um acesso direto aos diferentes componentes existentes na base de dados e uma implementação fácil e flexível das operações CRUD.

No entanto, trata-se de um procedimento que também apresenta as suas desvantagens, sendo uma delas a possibilidade de redução da *performance* quando são elaboradas *queries* complexas [106].

3.4.6. Django Rest Framework

Descrito como uma ferramenta poderosa e flexível para o desenvolvimento de *Web* APIs, o Django Rest Framework foi o escolhido para o desenvolvimento da REST API proposta, já que integra todos as benefícios e conveniências da *framework* Django e uma completa documentação de apoio, ao desenvolvedor. De entre as restantes *frameworks*, o DRF é uma das que revela um grau de maturidade maior, apresentado igualmente um elevado conjunto de funcionalidades para os desenvolvedores, permitindo a realização de boas práticas de desenvolvimento de serviços REST.

Esta framework é dotada de um conjunto de funcionalidades, que permite estender as caraterísticas das REST APIs, de uma forma simples e intuitiva. Dos aspetos diferenciadores possíveis de realçar desta *framework*, destaca-se a possibilidade de uma interface Web navegável da API, oferecendo uma grande usabilidade, simplicidade e compreensão por parte dos utilizadores.

Um dos pontos fortes da mesma, é a capacidade de construção do serviço, através de vistas baseadas em classes. Com isto, é possível o desenvolvimento da API, de um modo praticamente automático, através dos recursos disponíveis, não sendo necessário implementar a lógica a cada um dos *endpoints*[44] disponibilizados.

[44] Endereço ou ponto de conexão a um serviço *Web*

No entanto, existe a possibilidade de personalizar a lógica de negócio de cada uma das rotas.

O Django REST Framework, oferece uma poderosa capacidade de manipulação no modo como os recursos são apresentados, através do uso de serializadores, permitindo que haja negociação do formato em que os recursos são representados, e também a personalização de serializadores próprios, de forma a satisfazer as necessidades do serviço, a capacidade de autenticação de utilizadores, o controlo de acessos através de permissões e ainda capacidade de teste da API.

Por fim, oferece ainda a possibilidade de utilização de boas práticas da arquitetura REST de um modo simples, tais como paginação de conteúdos, utilização de filtros de conteúdos, entre outras. [107]

Arquitetura

Com base no padrão MVT presente na *framework* Django e na arquitetura REST que o Django REST Framework implementa, é possível de ser observado através da Figura 18 uma representação do funcionamento de uma REST API quando esta recebe um pedido HTTP do cliente.

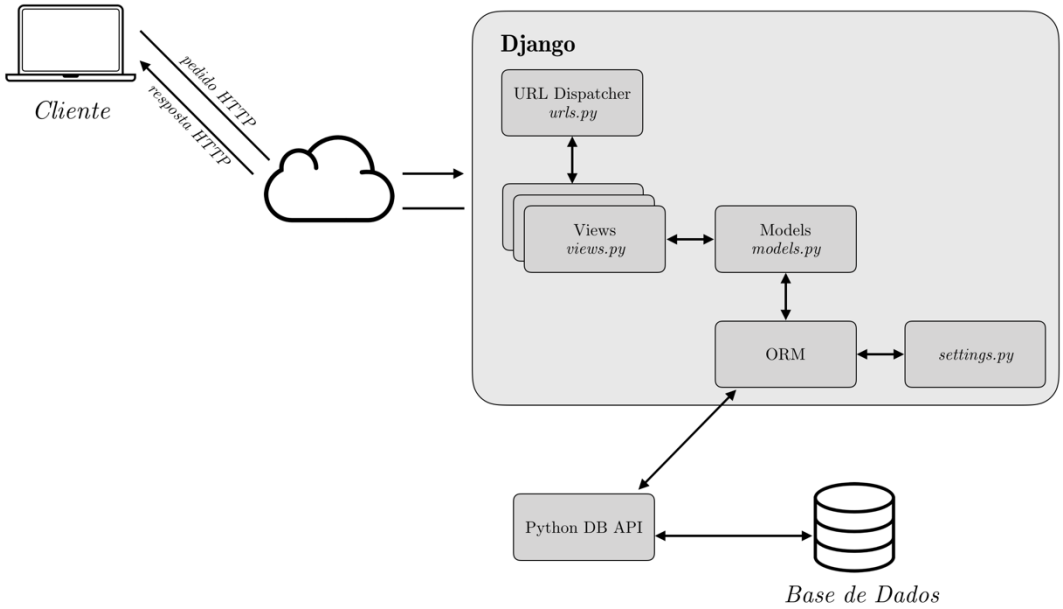


Figura 18 – Arquitetura de uma API desenvolvida pelo Django REST Framework

É também observável na Figura 18 os componentes que integram o projeto e como estes comunicam entre si, e de seguida é apresentado uma explicação detalhada do seu funcionamento:

- Quando um cliente realiza um pedido, o servidor *Web* recebe-o e através do mecanismo de rotas, converte-o para um URL específico.
- O distribuidor de URLs do Django, por sua vez e com base no ficheiro *urls.py,* seleciona o método da API correspondente ao URL solicitado, que será assim responsável por invocar a função de retorno - *view* - associada.
- Cada *view* retorna um objeto *HttpResponse* com a informação a apresentar ao cliente, na notação JSON, ou uma exceção HTTP. Para a recolha de dados provenientes da base de dados, a *view* recorre ao ficheiro *models.py*, onde é realizado o mapeamento, recorrendo ao padrão ORM, das tabelas que constituem a base de dados.
- O ORM recorre ao Python DB API, onde estão contidos um conjunto de módulos que permitem o acesso à base de dados. O acesso à base de dados é feito através objetos de conexão e cursores que são utilizados para operações de consulta e de registos na base de dados.

O ficheiro *settings.*py contém todas as configurações do projeto, nomeadamente informações relativas à conexão à base de dados, fuso horários, idioma e informações relativas à integração de *frameworks* no projeto.

3.4.7. Celery

O Celery[45], apresenta-se como um gestor de tarefas assíncronas, desenvolvido para integrar projetos desenvolvidos em Python, onde é possível dinamizar e automatizar parte da solução. Com o Celery, é possível executar uma fila de tarefas (recebidas através de mensagens), agendar tarefas diretamente no projeto, sem a necessidade de executáveis externos, apresentando uma fácil integração em *frameworks* Python, como é o caso do Django.

[45] http://www.celeryproject.org

A integração do Celery, servirá assim, para automatizar a tarefa de conexão, recolha e tratamento das publicações dos grupos de Facebook, evitando que a cada consulta à API, todo este processo seja realizado.

O *workflow* das tarefas assíncronas do Celery, conta com quatro atores:

- *Client*, também denominado por *Producer*, que é o responsável pela criação e gestão de *tasks*, sendo neste caso o Django.
- *Message Broker*, ou encaminhador de mensagens, é o responsável por gerir a fila de *tasks* que serão trocadas entre a solução e o Celery. Atualmente, os *message brokers* mais populares, devido à sua estabilidade e facilidade de integração são o RabbitMQ[46] e o Redis[47], tendo sido esta última a eleita para conduzir a troca de mensagens.
- *Worker*, ou *Consumer*, é o responsável por receber as tarefas do *message broker* e executá-las.
- *Result Store*, onde os *workers* podem ou não armazenar os resultados das tarefas que foram executadas.

Sucintamente, o *client* – o projeto Django –, pode passar uma tarefa, lista de tarefas, tarefas periódicas ou a repetição de uma tarefa para o *message broker* – Redis –. O *message broker* distribui essas tarefas entre os *workers* e, o resultado dessas tarefas pode ser escrito numa *result store* – base de dados da solução –, que mais tarde pode ser lido pelo *client* novamente. [108] [109]

[46] https://www.rabbitmq.com

[47] https://redis.io

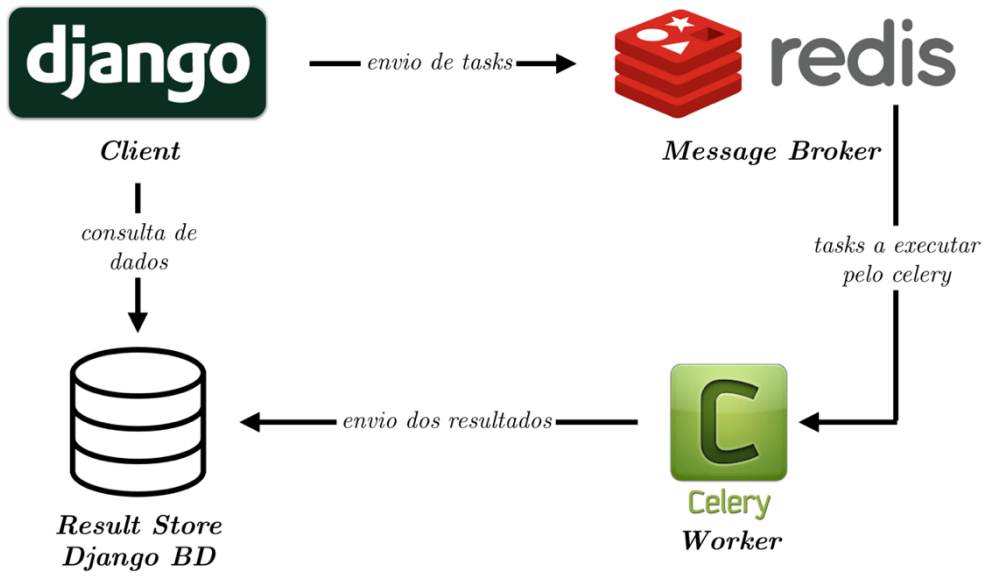


Figura 19 – Workflow do Celery adaptado: https://goo.gl/e2tMcD

3.5. Modelação dos Dados

A etapa relativa à interpretação e extração dos dados consideráveis proveitosos de integrar na solução, quando é realizado um pedido a um dos *providers*, é uma das etapas de maior importância no desenvolvimento da solução final.

De um modo geral, o retorno dos dados provenientes tanto das consultas realizadas através de métodos HTTP, como em SDKs, são expressos na notação JSON. Assim, e sabendo que o intuito é operar com várias plataformas existentes no mercado, o retorno dos dados que estas emitem são notoriamente diferentes. O intuito será então tratar essas informações, gerando um novo JSON que reúna um número de informações que se mantenham semelhantes entre as várias plataformas, de modo a que quando apresentada ao utilizador, este consiga comparar rapidamente qual a melhor oferta e escolher qual o serviço mais vantajoso.

Desta forma, existem elementos semelhantes que, independentemente do *provider*, estão presentes nas respostas que retornam. Desses, é possível apontar o preço da viagem, a distância e a duração da mesma. Todavia, e como os pedidos são realizados com a introdução das coordenadas – latitude e longitude – da posição de origem e destino é possível converter as mesmas na sua respetiva morada, caso esta não esteja presente na respetiva resposta do *provider*.

Outro campo fundamental que deve estar presente na resposta agregadora idealizada, é a data e hora referente à partida da viagem. Após o estudo dos *providers* e sabendo como estes arquitetam as suas respostas, nem todos disponibilizam diretamente a data e hora do início da viagem, relativa à hora em que foi realizado o pedido. Certos *providers*, disponibilizam o *datetime* relativo ao início da viagem diretamente na resposta; outros indicam o número de minutos estimados em que o motorista demora até se apresentar junto do cliente e, desta forma, é possível somar estes com a hora em que foi realizado o pedido.

O último parâmetro vital a estar presente na resposta, será o número de lugares livres em que o veículo destinado para a dada viagem, oferece. Desta forma, será possível segmentar de forma rápida, o volume de ofertas a apresentar ao cliente que procure uma viagem com um número mínimo de lugares livres.

As figuras seguintes, apresentam as respostas provenientes de cada um dos *providers* e, demonstram o que foi anteriormente discutido relativamente à heterogeneidade dos conteúdos presentes nas respostas providas por cada um destes. Desta forma, para as consultas realizadas, foi especificado como origem, a coordenada 41.158861,-8.630725 correspondente ao Edifício da Casa da Música da Cidade do Porto, e como destino, a coordenada 40.203264,-8.407545, que indica o Estádio Cidade de Coimbra, à exceção do *provider* Lyft, uma vez que ainda não opera em cidades portuguesas, e do Taxifare, que apresenta mais informações para cidades fora de Portugal, foram tomados em consideração dois pontos relativos à cidade de Boston, com as coordenadas de origem 42.356514,-71.05588, e as de destino, 42.348427,-71.072359.

```
[{'currency_code': 'EUR',
  'display_name': 'UberX',
  'distance': 76.94,
  'duration': 4920,
  'estimate': '€88-115',
  'high_estimate': 115.0,
  'localized_display_name': 'UberX',
  'low_estimate': 88.0,
  'minimum': 3,
  'product_id': '4dba36f8-e760-42f6-8da4-5515aaac629b
  'surge_multiplier': 1.0},
 {'currency_code': 'EUR',
  'display_name': 'Green',
  'distance': 76.94,
  'duration': 4920,
  'estimate': '€88-115',
  'high_estimate': 115.0,
  'localized_display_name': 'Green',
  'low_estimate': 88.0,
  'minimum': 3,
  'product_id': 'db207be9-306f-43b1-aa8e-b36a4cfb5b83
  'surge_multiplier': 1.0}]
```

```
[{'display_name': 'UberX',
  'estimate': 120,
  'localized_display_name': 'UberX',
  'product_id': '4dba36f8-e760-42f6-8da4-5515aaac629b'
 {'display_name': 'Green',
  'estimate': 180,
  'localized_display_name': 'Green',
  'product_id': 'db207be9-306f-43b1-aa8e-b36a4cfb5b83'
```

Figura 20 – Resposta de uma consulta realizada ao provider Uber
À esquerda: Estimativas de distância, duração e preço
À direita: Estimativa do tempo até se apresentar junto do cliente (em segundos)

```
"arrival_place": {
  "address": "Largo da Portagem, Largo da Portagem, Coimbra, Portugal",
  "city_name": "Coimbra",
  "country_code": "PT",
  "latitude": 40.20747,
  "longitude": -8.429643
},
"booking_mode": "auto",
"booking_type": "online",
"car": {
  "comfort": "Normal",
  "comfort_nb_star": 2,
  "id": "73996845",
  "make": "AUDI",
  "model": "A4"
},
"commission": {
  "currency": "EUR",
  "string_value": "1,50 €",
  "symbol": "€",
  "value": 1.5
},
"corridoring_type": "PLANNED",
"departure_date": "22/10/2018 14:40:00",
"departure_passenger_routing": {
  "proximity": "CLOSE",
  "routes": [
    {
      "distance_in_meters": 280,
      "type": "AS_THE_CROW_FLIES"
    }
  ]
},
"departure_place": {
  "address": "Casa da Música, Porto, Portugal",
  "city_name": "Porto",
  "country_code": "PT",
  "latitude": 41.160575,
  "longitude": -8.628282
},
"distance": {
  "unity": "km",
  "value": 121
},
"duration": {
  "unity": "s",
  "value": 4800
},
"freeway": true,
"is_comfort": true,
"links": {
  "_front": "https://www.blablacar.pt/trip-porto-coimbra-1270421373",
  "_self": "https://api.blablacar.pt/api/v2/trips/1270421373-porto-coimbra?_locale=pt_PT"
},
"locations_to_display": [
  "Porto",
  "Coimbra"
],
"permanent_id": "1270421373-porto-coimbra",
"price": {
  "currency": "EUR",
  "price_color": "BLACK",
  "string_value": "7 €",
  "symbol": "€",
  "value": 7
},
"price_with_commission": {
  "currency": "EUR",
  "price_color": "BLACK",
  "string_value": "8,50 €",
  "symbol": "€",
  "value": 8.5
},
```


Figura 21 – Resposta de uma consulta realizada ao provider BlaBlaCar

```
{
  "can_request_ride": true,
  "cost_token": null,
  "currency": "USD",
  "display_name": "Lyft",
  "estimated_cost_cents_max": 1000,
  "estimated_cost_cents_min": 800,
  "estimated_distance_miles": 1.48,
  "estimated_duration_seconds": 654,
  "is_valid_estimate": true,
  "price_quote_id": "ENnD6dvpLA==",
  "primetime_confirmation_token": null,
  "primetime_percentage": "0%",
  "ride_type": "lyft"
},
{
  "can_request_ride": true,
  "cost_token": null,
  "currency": "USD",
  "display_name": "Lyft XL",
  "estimated_cost_cents_max": 1500,
  "estimated_cost_cents_min": 1200,
  "estimated_distance_miles": 1.48,
  "estimated_duration_seconds": 654,
  "is_valid_estimate": true,
  "price_quote_id": "ENnD6dvpLA==",
  "primetime_confirmation_token": null,
  "primetime_percentage": "0%",
  "ride_type": "lyft_plus"
},
```

```
{
  "display_name": "Lyft",
  "eta_seconds": 120,
  "is_valid_estimate": true,
  "ride_type": "lyft"
},
{
  "display_name": "Lyft XL",
  "eta_seconds": 183,
  "is_valid_estimate": true,
  "ride_type": "lyft_plus"
},
```


Figura 22 – Resposta de uma consulta realizada à Lyft
À esquerda: Estimativas de distância, duração e preço
À direita: Estimativa do tempo até se apresentar junto do cliente (em segundos)


```
{
  "currency": {
    "int_symbol": "EUR"
  },
  "distance": 124315.2,
  "duration": 4591.5,
  "extra_charges": [],
  "flat_rates": [],
  "initial_fare": "3.25",
  "locale": "pt_PT",
  "metered_fare": 38.2,
  "rate_area": "Porto, Portugal",
  "status": "OK",
  "tip_amount": 0,
  "tip_percentage": 0,
  "total_fare": 41.45
}
```

```
{
  "businesses": [
    {
      "name": "Top Cab",
      "phone": "617-266-4800",
      "type": "taxi"
    },
    {
      "name": "Boston Airport Cab",
      "phone": "617-576-9800",
      "type": "taxi"
    },
    {
      "name": "Boston Cab",
      "phone": "617-536-5010",
      "type": "taxi"
    },
    {
      "name": "Metro Cab",
      "phone": "617-782-5500",
      "type": "taxi"
    }
  ],
  "status": "OK"
}
```


Figura 23 – Resposta de uma consulta realizada ao Taxifare
À esquerda: Estimativas de distância, duração e preço
À direita: Especificação dos serviços de táxis operáveis na zona

Como verificado através das figuras apresentadas, é possível notar a clara diferença entre os conteúdos retornados por cada *provider*, quando é realizada uma consulta aos seus serviços.

Contudo, podem existir outras informações, associadas a um dado *provider*, interessantes de integrar na resposta genérica arquitetada, que não estejam presentes nas demais, portanto, a presença de um campo livre que armazene estas, é um aspeto positivo a incluir.

A Listagem 1, apresenta assim, de seguida, a arquitetura da nova resposta JSON, que irá ser composta pelas informações dos *providers* anteriores apontados.

Listagem 1 – Formato de resposta proposto

```
{
  “uuid”: valor,
  “origin”: valor,
  “destination”: valor,
  “origin_name”: valor,
  “destination_name”: valor,
  “trip_duration”: valor,
  “trip_distance”: valor,
  “trip_datetime”: valor,
  “price”: valor,
  “price_currency”: valor,
  “free_seats”: valor,
  “provider”: valor,
  “provider_data”: {
    valor
   },
  “search_datetime”: valor
}
```

Com base nesta análise, foi possível proceder-se à primeira modelação do sistema, identificando assim quais os atributos relativos a uma viagem, necessários de armazenar numa base de dados.

Tabela 11 – Modelo relativo ao armazenamento das informações das viagens

Atributo	**Descrição**
uuid	Identificador Único e Universal da viagem
origin	Coordenada (Latitude, Longitude) relativa à posição de partida da viagem
destination	Coordenada (Latitude, Longitude) relativa à posição de destino da viagem
origin_name	Morada completa da coordenada relativa à posição de partida da viagem
destination_name	Morada completa da coordenada relativa à posição de destino da viagem
trip_duration	Estimativa da duração, em minutos, da viagem
trip_distance	Estimativa da distância, em metros, da viagem
trip_datetime	Estimativa da data e hora relativas ao início da viagem
price	Estimativa de preço da viagem
price_currency	Moeda relativa ao preço aplicado à viagem
free_seats	Número de assentos livres no automóvel para a viagem solicitada
provider	Nome do provedor do serviço
provider_data	Informações específicas de cada *provider* relativas à viagem requisitada
search_datetime	Data e hora relativas à consulta efetuada pelo utilizador

As viagens recolhidas através da rede social Facebook, não foram propositadamente apresentadas anteriormente, uma vez que as mesmas, como já apresentado, são recolhidas periodicamente através de um *crawler*, que irá recolher as informações presentes nas publicações de um dado grupo de boleias. Desta forma, é de igual modo expectável, que as viagens provenientes desta rede social, se façam

acompanhar das mesmas informações, como se tratasse de uma consulta a um *providers*, já apresentados.

Uma vez que é espectável a recolha de viagens de diferentes grupos de partilha de viagens, torna-se assim necessário o armazenamento dos dados relativos à identificação dos mesmos, para posterior conexão à Facebook Graph API e execução do *crawler*, de forma automática e periódica. A tabela seguinte apresenta esses mesmos atributos.

Tabela 12 – Modelo relativo ao armazenamento das informações dos grupos do Facebook

Atributo	**Descrição**
group_id	Identificador do grupo do Facebook
group_name	Nome do grupo do Facebook
group_keywords	Palavras chaves que caracterizem o grupo, como por exemplo, as cidades disponíveis para as partilhas de viagens

Para a conexão dos outros *providers* foi também planeado um *model* para o armazenamento de todas as credências responsáveis pela correta conexão aos serviços de cada *provider* incluído na solução, e a Tabela 13, apresentada de seguida, identifica assim os atributos para essa gestão.

A utilidade em guardar esta informação na base de dados, é a de evitar que esteja introduzida *hard-coded* no código da solução, evitando assim a perceção das mesmas, num possível ataque de dia zero.

Tabela 13 – Modelo relativo ao armazenamento das informações providers presentes na solução

Atributo	**Descrição**
description	Nome do *provider*
client_id	Identificação do ASMA perante um dado *provider*
client_secret	Chave secreta do ASMA perante um dado *provider*
token	*Access-token* do ASMA para a conexão aos serviços do *provider*

Por último, e como é esperada a utilização da solução por um número considerado de clientes, podendo estes apresentar um conjunto de privilégios distintos entre si, é utilizado o *model Users*, gerado automaticamente pela *framework* do Django, para o armazenamento e gestão dos dados relativos à identificação de utilizadores.

Este *default model*, atende bem às espectativas para um projeto de pequena/média escala, no que se refere à representatividade de utilizadores. Porém, e uma vez que o foco da solução, nesta fase piloto, é prover as informações das viagens, a gestão de utilizadores será concebida através deste *model* estandardizado.

Associado a cada utilizador, independentemente dos privilégios que este detenha, está o *access-token* atribuído para que seja possível a conexão à ASMA API e, consecutivamente a realização de pedidos.

Tabela 14 – Modelo relativo à associação dos access-tokens aos respetivos utilizadores

Atributo	**Descrição**
token	*Access-token* gerado aquando o registo de um novo utilizador na plataforma
owner	Instância do *model User*, referenciando um dado utilizador

A figura seguinte esquematiza os *models* anteriormente explanados e a relação entre os mesmos.

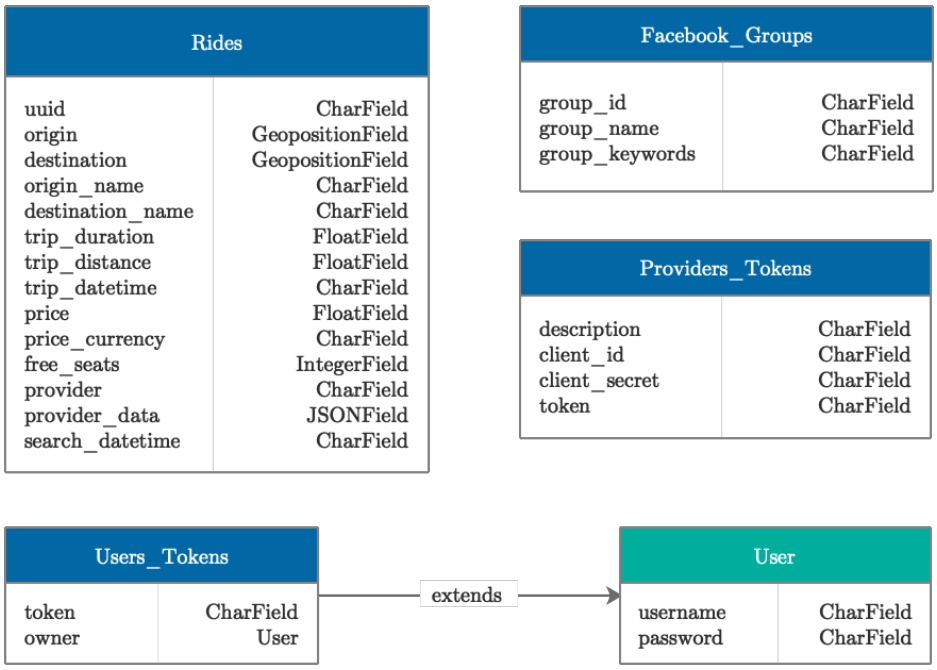


Figura 24 – Modelo de Dados

Por último, o sistema de Base de Dados escolhido para armazenar, estruturar e manipular os dados da solução foi a ferramenta nativa de armazenamento de dados de um projeto Django – o SQLite3[48] – modelado através dos Django Models.

O SQLite é assim um sistema de gestão de base de dados relacional de domínio público e multiplataforma, onde integra uma grande parte da linguagem SQL padrão, adicionando ainda alguns recursos próprios [110]. Ao contrário de outros SGBDs, o SQLite apresenta um design *serverless*, ou seja, não é um mecanismo de base de dados cliente-servidor, exigindo menos configuração que estes últimos [111]. Ao invés disso, o SQLite armazena a totalidade da base de dados (definições, tabelas, índices e os próprios valores) num único ficheiro na máquina *host*.

Por outro lado, os *Models* são assim caracterizados como o melhor recurso para persistir dados, e nestes é possível definir os campos (entenda-se atributos), e o seu tipo, como por exemplo *IntegerField*, para a definição de campos numéricos não decimais, e *CharField*, para campos que sejam compostos por uma cadeia de caracteres, semelhantemente às alternativas das bases de dados relacionais popularmente conhecidas.

Contrariamente, esta é definida como uma base de dados orientada a objetos, onde os *Models* não são modelados em tabelas, mas sim em classes Python e todas as consultas realizadas são através de comandos Python [112].

Os *Models* suportam um vasto conjunto de tipos possíveis de definir um atributo [113], porém, é possível empregar tipos de atributos desenvolvidos por terceiros e integrá-los na solução final do projeto. Exemplo disso é a utilização do GeopositionField, que permite armazenar uma coordenada, composta por latitude e longitude, e definir esse tipo de dados na tabela, entenda-se classe, pretendida.

[48] https://sqlite.org

CAPÍTULO **4**

Desenvolvimento da API

Neste capítulo serão apresentados diversos aspetos que foram importantes na conceção da API, tendo por base os requisitos descritos no Capítulo 3. Um dos pontos mais importantes no desenho de uma API REST é a representação e a modelação dos recursos e das relações entre eles. Precisamente por isso, neste capítulo é dada grande atenção aos recursos.

Como consequência da análise dos requisitos e dos casos de uso, foram identificados diversos recursos que a API tem que disponibilizar aos clientes. Uma vez que o propósito da dissertação, consistia no desenvolvimento da primeira versão da API, a deliberação dos recursos foram essencialmente definidos pelo Segundo Outorgante – Ubiwhere Lda.

Contudo, houve total liberdade para reformular e acrescentar novos recursos face aos propostos e autonomia para conceber a modelação necessária para o efeito.

4.1. Estrutura do Projeto

Como apresentado anteriormente o desenvolvimento da REST API, foi desenvolvida na linguagem de programação Python, com apoio da *framework* Django Rest Framework.

A estrutura do projeto é, em parte, gerada automaticamente aquando um projeto DRF, onde são criados um conjunto de diretórios e ficheiros (grande parte destes dedicados à configuração do projeto), como observável na Figura 25.

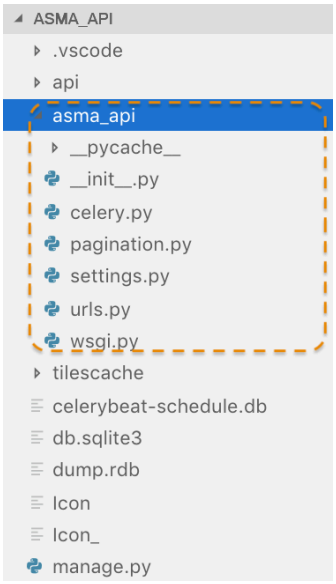


Figura 25 – Estrutura de pastas e ficheiros (ficheiros de configuração)

Na pasta *asma_api*, encontram-se sobretudo os ficheiros de configuração da solução, onde é possível indicar:

- *settings.py* – ficheiro de configuração do projeto;
- *urls.py* – ficheiro de configuração de URLs, onde é expresso o mapeamento de cada URL para a *view* correspondente.

```
urlpatterns = [
    url(r'api/docs/$', schema_view.with_ui('swagger', cache_timeout=0), name='schema-swagger-ui'),

    url(r'api/rides/$', api.core_processor.rides_methods.GetRides.as_view(), name='Get-Rides'),
    url(r'api/rides/uber/$', api.core_processor.uber_methods.UberView.as_view(), name='UBER-View'),
    url(r'api/rides/blablacar/$', api.core_processor.blablacar_methods.BlaBlaCarView.as_view(), name='BlaBlaCar-View'),
    url(r'api/rides/lyft/$', api.core_processor.lyft_methods.LyftView.as_view(), name='Lyft-View'),
    url(r'api/rides/taxifare/$', api.core_processor.taxifare_methods.TaxifareView.as_view(), name='TaxiFare-View'),
    url(r'api/rides/facebook/$', api.core_processor.facebook_methods.FacebookView.as_view(), name='Facebook-view'),
]
```

Figura 26 – Configuração das rotas, presentes no ficheiro urls.py

E também configurações adicionais *à posteriori*, relativas à integração de outras frameworks ou de configurações pertinentes para o enriquecimento da solução.

- *celery.py* – ficheiro que contém configurações relativas ao gestor de tarefas assíncronas, Celery.
- *pagination.py* – ficheiro de configuração onde é expresso o número de objetos JSON, a retornar na resposta, e os respetivos elementos.

```python
class SmallPagesPagination(PageNumberPagination):
    page_size = 5
    page_size_query_param = 'limit'

    def get_paginated_response(self, data):
        return Response(OrderedDict([
            ('current_page', self.page.number),
            ('total_results', self.page.paginator.count),
            ('total_pages', self.page.paginator.num_pages),
            ('next_page', self.get_next_link()),
            ('previous_page', self.get_previous_link()),
            ('results', data)
        ]))
```

Figura 27 – Ficheiro de configuração pagination.py

Os restantes diretórios são relativos a outras funcionalidades do projeto, como o *facebook_crawler* onde está implementado todo o processo de conexão à Graph API, extração e processamento dos dados contidos nas publicações dos grupos onde é realizado o *crawler*, como também o processo relativo ao armazenamento dos recolhidos (já tratados) na base de dados; *migrations* como o diretório responsável por manter um sistema de controlo de versões da base de dados; o diretório *static*, onde se encontram os ficheiros considerados como estáticos, podendo estes ser imagens, ficheiros javascript ou css; *template* que é normalmente composto por ficheiros HTML, com o objetivo de representar visualmente os dados; e por último, o diretório *utils*, criado para agregar um conjunto de ficheiros que dão suporte e realizam ações para um correto funcionamento da ASMA API.

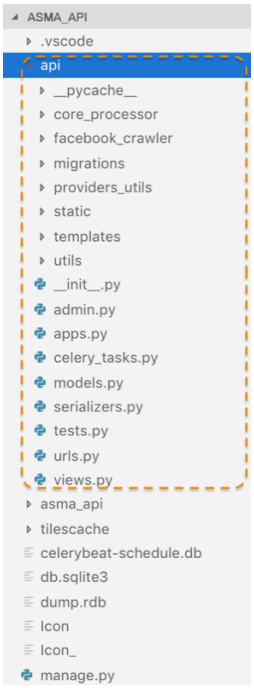


Figura 28 – Estrutura de pastas e ficheiros (ficheiros de desenvolvimento)

4.2. Fluxo da Solução

Como indicado anteriormente, o principal método de interação com uma API é a partir de pedidos HTTP, especificando um dos métodos aos *endpoints* desenvolvidos.

Os *endpoints* desenvolvidos que permitem a obtenção de dados são seis, todos com propósitos diferentes, e apresentados na seguinte tabela, ilustrando o método GET.

Tabela 15 – Descrição dos principais endpoints da solução

Endpoint	**Descrição**
/rides	Consulta das viagens disponíveis em todos os *providers* presentes na solução
/rides/uber	Consulta das viagens disponíveis para o *provider* Uber
/rides/blablacar	Consulta das viagens disponíveis para o *provider* BlaBlaCar
/rides/lyft	Consulta das viagens disponíveis para o *provider* Lyft
/rides/taxifare	Consulta das viagens disponíveis para o *provider* TaxiFare
/rides/facebook	Consulta das viagens extraídas dos grupos do Facebook

Porém, para realizar um pedido corretamente é necessário especificar no URL do pedido, os parâmetros para concretizar o mesmo. Destes, é possível apontar os obrigatórios, como a origem e o destino da viagem pretendida, e os opcionais, que organizam ou filtram a coleção dos dados retornados na resposta. Desta forma, os parâmetros possíveis de introduzir são:

Tabela 16 – Parâmetros possíveis de indicar no URL do pedido

Parâmetros		**Obrigatoriedade**	**Descrição**
origin		Obrigatório para o retorno de uma nova coleção de viagens	Coordenada geográfica relativa a posição de origem e destino da viagem
destination			
uuid		Obrigatório para o retorno de uma viagem já pesquisada	Identificação de uma viagem, armazenada na BD, já pesquisada
order_by	price_asc	Não Obrigatório	Ordenação das viagens pela ordem ascendente do preço
	price_des	Não Obrigatório	Ordenação das viagens pela ordem descendente do preço
	uber	Não Obrigatório	O *provider* Uber aparecerá em primeiro nos resultados
	blablacar	Não Obrigatório	O *provider* BlaBlaCar aparecerá em primeiro nos resultados
	lyft	Não Obrigatório	O *provider* Lyft aparecerá em primeiro nos resultados
	taxifare	Não Obrigatório	O *provider* TaxiFare aparecerá em primeiro nos resultados
	facebook	Não Obrigatório	O *provider* Facebook aparecerá em primeiro nos resultados
free_seats		Não Obrigatório	Indicação do número de lugares pretendidos para a viagem

Apresentados assim os *endpoints* e os parâmetros, é possível indicar um esqueleto genérico da anatomia dos URLs. E o mesmo é apresentado de seguida.

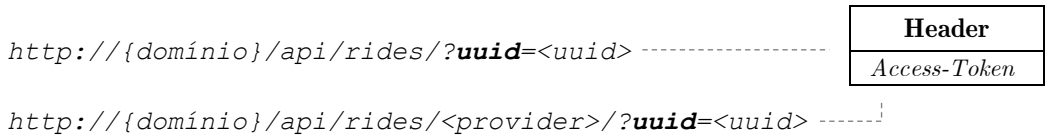
Tabela 17 – Anatomia dos URLs de pesquisa

URL	Propósito
http://{domínio}/api/**rides**/?{parâmetros obrigatórios}&{parâmetros opcionais}	Consulta das viagens disponíveis em todos os *providers* presentes na solução
http://{domínio}/api/**rides/uber**/?{parâmetros obrigatórios}&{parâmetros opcionais}	Consulta das viagens disponíveis para o *provider* Uber
http://{domínio}/api/**rides/blablacar**/?{parâmetros obrigatórios}&{parâmetros opcionais}	Consulta das viagens disponíveis para o *provider* BlaBlaCar
http://{domínio}/api/**rides/lyft**/?{parâmetros obrigatórios}&{parâmetros opcionais}	Consulta das viagens disponíveis para o *provider* Lyft
http://{domínio}/api/**rides/taxifare**/?{parâmetros obrigatórios}&{parâmetros opcionais}	Consulta das viagens disponíveis para o *provider* Taxifare
http://{domínio}/api/**rides/facebook**/?{parâmetros obrigatórios}&{parâmetros opcionais}	Consulta das viagens extraídas dos grupos do Facebook

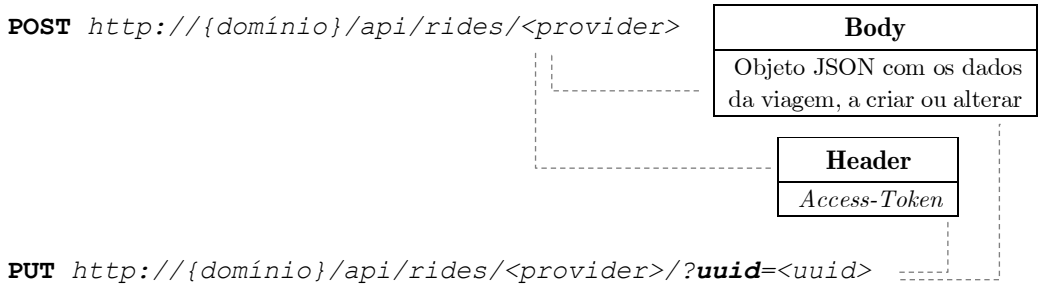
De notar que, o método GET não foi o único implementado. Foram de igual modo implementados os métodos DELETE, POST, PUT, por forma a realizar as operações CRUD à API desenvolvida.

Primeiro, convém notar também, que todas as consultas realizas à API, independentemente do método têm de ser acompanhadas por um *access-token*, no cabeçalho do pedido, para validar a autenticidade do utilizador, conforme apresentado no Capítulo 3 – Levantamento de Requisitos.

O método **DELETE**, pode ser utilizado em qualquer dos *endpoints* apresentados na Tabela 15. A consulta, para além do *access-token*, deve ser acompanhada do uuid da viagem que se pretende remover. Assim, é possível a remoção de viagens através da utilização de um dos seguintes URLs, apresentados abaixo. Na eventualidade de desejar remover uma viagem utilizando o URL onde é especificado um *provider*, existe uma validação que verifica se o *provider* relativo ao *uuid* introduzido no URL corresponde ao armazenado na BD; caso não a remoção não será aplicada e será retornada uma resposta HTTP 400 – Bad Resquest –, que informará que a viagem identificada, não corresponde ao *provider* especificado.



Os restantes métodos, o **POST** e o **PUT**, têm como finalidade a criação de uma nova viagem ou a alteração de uma já existente e armazenada na BD, respetivamente. Ao contrário dos métodos anteriores, a especificação dos campos, quer para a definição de uma nova viagem, quer para a alteração de uma viagem já armazenada na base de dados é através da introdução de um objeto JSON no campo *body* do pedido HTTP, com os respetivos dados da viagem. Com isto, é minimizado o comprimento do URL de pedido, como também não são expostos os campos que contêm informações confidências relativamente às viagens. Para além disto, ambos os métodos têm de ser acompanhados, no cabeçalho HTTP, de um *access-token* válido, de modo a permitir realizar tais operações, e só são admissíveis consultas aos URLs que tenham especificado o *provider* para qual se pretende criar ou editar uma viagem. De notar que, para o método PUT, o URL do pedido tem ser acompanhado com o *uuid* da viagem que se pretende alterar.



Além dos *endpoints* apresentados, foi igualmente criado um que direciona o utilizador para uma página onde está detalhada toda a documentação da API desenvolvida. Esta é possível através da integração da *framework* Swagger[49], de modo a permitir o acesso à documentação de uma forma fácil e intuitiva, auxiliando assim um desenvolvedor a interagir com a solução.

De notar que, para consultar a página da documentação basta aceder ao URL `http://{domínio}/api/docs`, pelo *browser*, sem a necessidade de ser introduzido algum *access-token* para permitir a navegação pela mesma. Ao consultar o Anexo A – Documentação da API, são apresentados mais exemplos relativos à documentação da API elaborada.

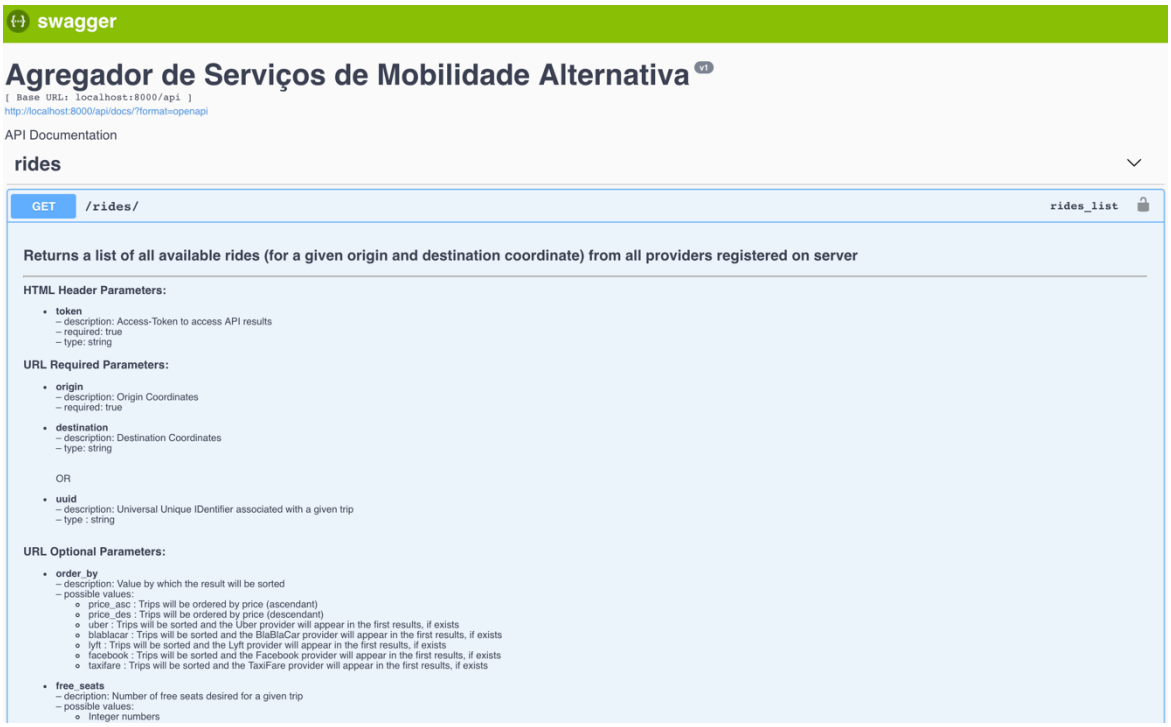


Figura 29 – Exemplo da framework Swagger

Apresentados os detalhes que permitem a correta conexão à API, os passos descritos de seguida, mostram o fluxo de atividades quando é realizado um pedido.

Como já apresentado, cada *endpoint* criado é representado em classes Python, sendo nestas que serão desenvolvidas todas as ações referentes à procura de viagens, ao armazenamento destas na base de dados e ao retorno de uma coleção de viagens numa resposta JSON.

[49] https://swagger.io

Tomando como exemplo um pedido GET ao *endpoint* `/rides`, serão iterados todos os *providers*, e averiguada a existência de viagens disponíveis para o par de coordenadas especificado.

Desta forma, o pedido seguinte representa uma consulta ao *endpoint* `/rides`, onde é especificado como origem, a coordenada 41.158861,-8.630725 correspondente ao Edifício da Casa da Música da Cidade do Porto, e como destino a coordenada 40.203264,-8.407545, que indica o Estádio Cidade de Coimbra.

*http://{domínio}/api/rides/?**origin**=41.158861,-8.630725&**destination**=40.203264,-8.407545*

Header
Access-Token

O interpretador presente no Django Rest Framework, automaticamente deteta se o *endpoint* presente no URL está contido na lista de *urls* definidos no ficheiro *urls.py* – Figura 26 – e, no caso de se verificar, é iniciado o código relativo à classe em que o determinado *endpoint* está associado.

Assim, e para o URL apresentado anteriormente, são realizados os seguintes passos:

- Verificação se o *Header* do pedido contém um *access-token* inserido. Caso contrário, é retornada uma resposta com o código 400, com a descrição "*Provide your access-token on the Header*";
- Verificação se o *access-token* introduzido no *Header* do pedido, existe na base de dados. Caso não exista, é retornada uma resposta com o código 400, com a descrição "*Invalid Access-Token*";
- Se forem verificados, com sucesso, os passos anteriores, é recolhida a data e hora, de modo a preservar o instante em que a consulta foi realizada;
- Seguidamente, são extraídos os parâmetros obrigatórios de uma consulta, a origem – *origin* – e o destino – *destination* – e avaliado se a introdução das mesmas obedece às do padrão XX.XXXXXX. Caso não obedeça, é retornada uma resposta com o código 400, com a descrição "*Inserted Coordinates are Invalid. Use the following pattern XX.XXXXX,XX.XXXXX*";

- Quando os parâmetros *origin* e *destination* são recolhidos é realizada a conversão das coordenadas geográficas na sua respetiva morada, onde está presente o nome da rua, o código-postal, a cidade e o país, através da biblioteca GeoCoders – apresentada no capítulo 4.7 – Bibliotecas Utilizadas. Na eventualidade dos serviços da biblioteca estarem indisponíveis, ou atingido o número de pedidos autorizados por dia, é retornada uma resposta com o código 400, com a descrição "*Geo Converter Temporarily Down. Repeat your request again*";
- Verificado assim o sucesso das etapas anteriores, o próximo passo é a conexão aos serviços dos *providers*, a recolha das informações, se existentes, e o armazenamento das mesmas na base de dados. O processo de conexão e recolha de viagens é bastante idêntico entre os *providers*, contudo, existem pontuais diferenças, sendo estas apresentadas no tópico 4.3 – Recolha dos dados dos *providers*.
- Encontradas assim viagens, e estando as informações das mesmas já armazenadas na base de dados do sistema, o próximo passo é relativo à serialização dos dados, onde é realizada uma consulta à base de dados e, seguidamente, retornada a resposta para o utilizador, com o código 200 e com a coleção de viagens disponíveis para o instante da pesquisa. Uma vez que o URL do pedido não inclui nenhum dos parâmetros opcionais, a consulta à base de dados, de modo a recolher as viagens armazenadas, irá proceder-se apenas com os campos da origem e destino recolhidos do URL do pedido, bem como a data e hora em que a pesquisa foi realizada.

  Recolhidas assim o conjunto de viagens, o próximo passo e último passo, é o envio das mesmas ao utilizador, numa resposta JSON com suporte a paginação, conforme observável na Figura 30.

```
uber_serializer = RidesSerializer(Rides.objects.filter(origin = origin_inserted, destination = destination_inserted, \
                search_datetime = search_datetime, provider = "UBER"), many=True).data

blablacar_serializer = RidesSerializer(Rides.objects.filter(origin = origin_inserted, destination = destination_inserted, \
                search_datetime = search_datetime, provider = "BLABLACAR"), many=True).data

lyft_serializer = RidesSerializer(Rides.objects.filter(origin = origin_inserted, destination = destination_inserted, \
                search_datetime = search_datetime, provider = "LYFT"), many=True).data

taxifare_serializer = RidesSerializer(Rides.objects.filter(origin = origin_inserted, destination = destination_inserted, \
                search_datetime = search_datetime, provider = "TAXIFARE"), many=True).data

facebook_serializer = RidesSerializer(Rides.objects.filter(origin_name = origin_city, destination_name = destination_city, \
                    provider = "FACEBOOK"), many=True).data

final_serializer = uber_serializer + blablacar_serializer + lyft_serializer + taxifare_serializer + facebook_serializer

page = self.paginate_queryset(final_serializer)
return self.get_paginated_response(page)
```

Figura 30 – Recolha e Serialização das Viagens

No entanto, e como apresentado anteriormente, uma consulta pode ser acompanhada com parâmetros opcionais, podendo tanto filtrar os conteúdos a apresentar, ou organizar os mesmos.

Tomando desta vez em consideração o seguinte URL, com a mesma origem e destino do apresentado anteriormente, mas com a indicação do parâmetro opcional *order_by*, as viagens a retornar ao utilizador, serão apresentadas estando as mesmas ordenadas de modo ascendente, o que significa, que as viagens estarão dispostas da mais acessível, até à mais cara.

*http://{domínio}/api/rides/?**origin**=41.158861,-8.630725*
*&**destination**=40.203264,-8.407545&**order_by**=price_asc*

Header
Access-Token

Os passos para se atingir este fim, são os mesmo que foram apresentados anteriormente, contudo a consulta realizada, no último ponto, à base de dados, é ligeiramente diferente.

Tomando como exemplo o caso do Facebook – descrito em 4.4 - Análise e Processamento de Linguagem Natural em Redes Sociais –, é notável na Figura 31, que existem duas consultas a esse *provider*. A primeira, é para assegurar o retorno de viagens com um preço maior ou igual a 0€, e a segunda, para retornar as viagens que não apresentam um preço especificado pelo autor da publicação, internamente definido com o preço de "-1". Esta última, não entrará na concatenação dos dados retornados da base de dados, uma vez que foi definido que as publicações onde não é definido um preço, aparecerão no final da coleção de viagens a apresentar ao utilizador.

Para se proceder à ordenação da lista de viagens recolhidas, é aplicada uma expressão lambda de ordenação, que tomará em consideração o campo *price* de todas as viagens e ordenará o conjunto, neste caso, de modo ascendente.

Por último, e estando já ordenado o conjunto de viagens, o próximo e último passo, é o envio das mesmas ao utilizador, numa resposta JSON com suporte a paginação.

```
elif order_by_inserted == "price_asc":
    uber_queryset = Rides.objects.filter(origin = origin_inserted, destination = destination_inserted, \
                search_datetime = search_datetime, provider = "UBER")

    blablacar_queryset = Rides.objects.filter(origin = origin_inserted, destination = destination_inserted, \
                search_datetime = search_datetime, provider = "BLABLACAR")

    lyft_queryset = Rides.objects.filter(origin = origin_inserted, destination = destination_inserted, \
                search_datetime = search_datetime, provider = "LYFT")

    facebook_queryset1 = Rides.objects.filter(origin_name = origin_city, destination_name = destination_city, \
                provider = "FACEBOOK", price__gte=0)

    facebook_queryset2 = Rides.objects.filter(origin_name = origin_city, destination_name = destination_city, \
                provider = "FACEBOOK", price=-1)

    taxifare_queryset = Rides.objects.filter(origin = origin_inserted, destination = destination_inserted, \
                search_datetime = search_datetime, provider = "TAXIFARE")

    queryset_list = RidesSerializer(uber_queryset, many=True).data + \
                    RidesSerializer(blablacar_queryset, many=True).data + \
                    RidesSerializer(lyft_queryset, many=True).data + \
                    RidesSerializer(taxifare_queryset, many=True).data + \
                    RidesSerializer(facebook_queryset1, many=True).data

    partial_serializer = sorted(queryset_list, key=lambda x: x['price'])

    partial_serializer2 = sorted(RidesSerializer(facebook_queryset2, many=True).data, key=lambda x: x['price'])

    sorted_results = partial_serializer + partial_serializer2

page = self.paginate_queryset(sorted_results)
return self.get_paginated_response(page)
```

Figura 31 – Recolha e Serialização das Viagens, ordenadas pelo preço de modo ascendente

No que diz respeito à filtragem de viagens, relativa ao número de lugares livres que estas oferecem, o processo de recolha das mesmas é efetuado através da *query* realizada à base de dados, onde o campo *free_seats* for maior ou igual ao especificado no URL do pedido.

```
elif origin_inserted and destination_inserted and free_seats_inserted:
    if free_seats_inserted is not None and free_seats_inserted.isdigit():
        try:
            query = Rides.objects.filter(origin = origin_inserted, destination = destination_inserted, \
                search_datetime = search_datetime, free_seats__gte = free_seats_inserted)

            query_facebook = Rides.objects.filter(origin_name = origin_city, destination_name = destination_city, \
                provider = "FACEBOOK", free_seats__gte = free_seats_inserted)

            final_query = query + query_facebook
            serializer = RidesSerializer(final_query, many = True).data

            page = self.paginate_queryset(serializer)
            return self.get_paginated_response(page)
        except Exception:
            return Response(status=status.HTTP_400_BAD_REQUEST, data='Invalid Order By Clause')
```

Figura 32 – Recolha e Serialização das Viagens, filtrado pelo número de lugares livres

4.3. Recolha dos Dados dos *providers*

A conexão aos serviços dos *providers*, a recolha das informações e o armazenamento destas na base de dados, são sem dúvida passos cruciais para se atingir o completo sucesso dos passos anteriormente explanados no tópico 4.2 - Fluxo da Solução. Como já mencionado, o processo de conexão e de recolha de viagens é bastante idêntico entre os *providers*, havendo pontuais diferenças. Desta forma, são apresentados, de seguida, os casos do Uber e da BlaBlaCar, para notar as diferenças entre a conexão e a recolha dos dados provenientes de um *provider*, onde o processo é realizado através de um SDK ou por métodos HTTP a um *endpoint Web*.

Por sua vez, a extração das viagens do Facebook é o processo mais diferente e complexo presente na solução. Dessa forma, a explicação das etapas desenvolvidas relativamente à rede social é apresentada no tópico 4.4 - Análise e Processamento de Linguagem Natural em Redes Sociais.

Uber

Após corretamente instalado o SDK da Uber [114], é necessário estabelecer uma sessão com os serviços Uber e instanciar um cliente com o *token* que foi atribuído aquando do registo na plataforma de desenvolvedores, conforme exemplificado abaixo:

```
from uber_rides.session import Session
from uber_rides.client import UberRidesClient

session = Session(server_token = Providers_Tokens.objects.get(description = "uber_token").token)
client = UberRidesClient(session)
```

Figura 33 – Conexão aos serviços Uber

Seguidamente, e para que seja possível a consulta das estatísticas das viagens, é utilizado o método *get_price_estimates*, introduzindo tanto a latitude como a longitude da posição de origem e de destino. Com a resposta desta consulta, será então possível a recolha dos dados e a sua manipulação, para os armazenar, já tratados, na base de dados.

Porém, existem aspetos particulares, interessantes de apontar, como:

- A hora de partida da viagem é obtida através da adição dos minutos que o motorista demora até se apresentar ao passageiro, com a hora do pedido à API da Uber
- O número de lugares por viatura, é um dado que não vem expresso na resposta, quando é realizada uma consulta à Uber, pelo que foi definido um dicionário com os tipos de veículos oferecidos pela companhia e o número de lugares dos mesmos.

```
car_type = [li['display_name'] for li in uber_response]
duration = [li['duration'] / 60 for li in uber_response]
distance = [li['distance'] for li in uber_response]
product_id = [li['product_id'] for li in uber_response]
low_estimate = [li['low_estimate'] for li in uber_response]
high_estimate = [li['high_estimate'] for li in uber_response]
estimate = [li['estimate'] for li in uber_response]

if any('surge_multiplier' in d for d in uber_response):
    multiplier = [li['surge_multiplier'] for li in uber_response]
else:
    multiplier = [None] * dic_length

for i in range(0, n_offers):
    date_time.append(
        DateTimeCreator.convert_datetime_to_string(
            now.shift(minutes = get_ETA(origin_parts[0], origin_parts[1])[i])
        )
    )
```

```
for i in range(0, n_offers):
    bd = Rides()
    bd.provider = "UBER"
    bd.uuid = uuid.uuid4()
    bd.origin = Geoposition(origin_parts[0], origin_parts[1])
    bd.origin_name = origin_address
    bd.trip_datetime = date_time[i]
    bd.destination = Geoposition(destination_parts[0], destination_parts[1])
    bd.destination_name = destination_address
    bd.trip_duration = round(duration[i], 2)
    bd.trip_distance = round(distance[i], 2)
    bd.price = round(((low_estimate[i] + high_estimate[i]) / 2), 2)
    bd.free_seats = uber_seats.get(car_type[i].lower().replace(" ", ""))
    bd.search_datetime = search_datetime
    bd.provider_data = dict(
        {
            "car_type":car_type[i],
            "estimated_min_price": round(low_estimate[i], 2),
            "estimated_max_price": round(high_estimate[i], 2),
            "multiplier": multiplier[i],
            "trip_id": product_id[i]
        }
    )
    bd.save()
```

Figura 34 – Extração e manipulação dos dados provenientes da Uber

BlaBlaCar

Contrariamente ao anterior, a conexão aos serviços BlaBlaCar é realizada através de métodos HTTP e, para tal, é utilizado o módulo *client* da biblioteca *http*.

Para se realizar então a conexão, é necessário realizar uma instância do HTTPConnection, que representa uma transação para um servidor HTTP, e indicar o URL base para onde se deseja realizar o pedido. Seguidamente, e através do método *request*, é possível especificar o método HTTP, o *endpoint* e os parâmetros, juntamente com cabeçalho, se necessário, afim de realizar o pedido aos serviços BlaBlaCar.

```
import http.client

conn = http.client.HTTPSConnection("public-api.blablacar.com")

headers = {
    'accept': "application/json",
    'key': Providers_Tokens.objects.get(description="blablacar_token").token
}
conn.request("GET", "/api/v2/trips?"
                "fc=" + start_latitude + "," + start_longitude +
                "&tc="+ end_latitude + "," + end_longitude +
                "&locale=pt_PT"
                "&_format=json"
                "&cur=EUR"
                "&db=" + DateTimeCreator.get_string_formatted_date(now)
                "&sort=trip_date"
                "&order=asc" , headers=headers)

return conn.getresponse()
```

Figura 35 – Conexão aos serviços BlaBlaCar

De um modo semelhante ao anterior, as informações das viagens são recolhidas através da resposta à consulta da API da BlaBlaCar, conforme apresentado na Figura 36, e seguidamente armazenado na Base de Dados.

```
data = response.read().decode()
json_obj = json.loads(data, encoding='latin-1')
json_formatted = json.dumps(json_obj, sort_keys=True, indent=2, ensure_ascii=False)


trip = json_obj.get('trips')
number_offers = len(trip)

for i in trip:
    origin_name.append(i.get('departure_place').get('address'))
    origin_lat.append(i.get('departure_place').get('latitude'))
    origin_lon.append(i.get('departure_place').get('longitude'))

    origin_date.append(i.get('departure_date').split(" ")[0])
    origin_hour.append(i.get('departure_date').split(" ")[1][:8])
    destination_name.append(i.get('arrival_place').get('address'))
    destination_lat.append(i.get('arrival_place').get('latitude'))
    destination_lon.append(i.get('arrival_place').get('longitude'))

    price.append(i.get('price_with_commission').get('value'))
    distance.append(i.get('distance').get('value'))

    duration.append(i.get('duration').get('value')/60)
    trip_id.append(i.get('trip_details_id'))
    free_seats.append(i.get('seats_left'))

    if i.get('car') is not None:
        car_info.append( dict(i.get('car')) )
    else:
        car_info.append(None)

for i in range(0, len(origin_date)):
    datetime.append(
        DateTimeCreator.get_formatted_datetime_given_date_time(origin_date[i], origin_hour[i])
    )
```

Figura 36 – Extração e manipulação dos dados provenientes da BlaBlaCar

4.4. Análise e Processamento de Linguagem Natural em Redes Sociais

O presente tópico é reservado à extração de informações relativas à partilha de boleias através da rede social Facebook. Neste, são também apresentados conceitos como *crawlers*, técnicas de processamento de linguagem natural, como tokenização e uso de expressões regulares, e processos de automatização de tarefas periódicas.

Para auxiliar a leitura, estão disponíveis fluxogramas de modo a condensar e a sumariar as etapas subjacentes a um dado processo.

Como referido anteriormente, um dos objetivos contidos na proposta da dissertação, era a extração de dados relativos à partilha de viagens publicadas em redes sociais, como é o caso do Facebook.

Assim, e para que seja possível modelar os dados contidos nas publicações do Facebook numa resposta JSON, as informações essenciais a tomar em consideração são: nome e ID do utilizador, corpo da publicação, bem como, a data associada à mesma.

Quando esta informação tiver sido então recolhida, é dado início ao tratamento dos dados, onde estes terão de percorrer um conjunto de etapas até ser atingido o final esperado, e estas são:

- Substituição dos parágrafos (`\n`) por um ponto final (`.`), para facilitar o processo de tokenização.

- Carregamento de uma lista das cidades de Portugal, para uma posterior verificação se as cidades introduzidas na publicação realmente existem.

- Averiguação se a publicação recolhida se trata de um pedido ou de uma oferta de boleia. É avaliado em cada *post* recolhido a existência de palavras-chave ou de caracteres-chave relativos ao pedido de um serviço, como é o caso do "?", "procuro", "preciso" e "alguém". Esta etapa é alcançada através do processo de tokenização presente na biblioteca NLTK.

A tokenização permite a decomposição automática e inteligente, o documento em cada termo que o compõe. Os delimitadores utilizados para o processo de tokenização são o espaço, quebras de linha (parágrafos), tabulações e alguns caracteres especiais. Depois de decompostos, cada elemento é armazenado numa estrutura de lista, na qual é possível iterar e avaliar a presença dos elementos, mencionados anteriormente, que correspondem a um pedido de boleia.

Ao verificar-se assim a ocorrência de pelo menos um destes, é descartada a entrada associada da lista de publicações recolhidas, bem como toda a informação associada à mesma, como a data de publicação, *link* e os dados do autor, uma vez que o foco do *crawler* se prende na extração de ofertas de boleias e não no pedido das mesmas.

Tabela 18 – Exemplos de publicações consideradas como inválidas (procura de boleias)

Procuro boleia Aveiro-Porto
Preciso de uma boleia de Aveiro para Lisboa, às 15h
Alguém viaja hoje do Porto para Coimbra**?**

- O passo seguinte consiste em avaliar a data em que a publicação foi submetida, e várias etapas foram tomadas em consideração:
  - Primeiro, é avaliado se no corpo da publicação há referência a expressões que representam um futuro próximo, como a presença da palavra "hoje" ou "amanhã". Na eventualidade de a palavra "hoje" se verificar, é comparada a data atual com a data da publicação e, se ambas forem iguais, a publicação é considerada como válida e armazenada para posterior análise. Caso contrário, toda a informação associada à publicação é descartada. O mesmo acontece com a palavra "amanhã", porém a verificação efetuada é entre a soma do dia atual com o valor 1 e a soma da data da publicação com o valor 1, e se coincidirem, a publicação é considerada como válida.
  - As próximas verificações são relativas aos dias da semana, na eventualidade de um utilizador especificar um dia da semana concreto, como por exemplo, "segunda-feira".

Para isto, são tomados em consideração possíveis variações da mesma, frequentemente aplicada no dia-a-dia, como "segunda", "2f", "2feira", "2ªfeira" e "2ºfeira", considerando também a possibilidade de estes serem expressos com caracteres maiúsculos.

Quando se verifica esta situação, de o utilizador especificar um dia da semana em concreto, é aplicada uma nova verificação, com o objetivo de avaliar se a diferença entre a data atual não é superior a 3 dias. Com isto, são descartadas publicações cuja data de publicação supera os 3 dias, garantindo assim que apenas as publicações mais recentes sejam processadas. Para esta tomada de decisão, foi considerado o estudo de várias publicações em alguns grupos de partilha, e analisado o intervalo temporal a que as mesmas se referem. Ou seja, a maior concentração de publicações é relativa ao presente dia ou ao dia seguinte. Pontualmente, são também criadas publicações a uma sexta-feira, oferecendo viagens para segunda, daí o *range* temporal aceite, ser de 3 dias. Assim, se verificado que o intervalo temporal não excede os 3 dias, mais dois novos casos são verificados.

  - Para os casos, "Boleia Porto-Aveiro, sexta dia 10" ou "Boleira Porto-Aveiro 20/07", foi desenvolvido um algoritmo que, tomando em consideração o dia da semana atual com o dia da semana especificado na publicação, retorna a data (dia, mês e ano) do dia pretendido. Tomando como exemplo o dia da semana Quarta-Feira, e o dia da boleia como Sexta-Feira, é calculada a distância entre estes dois dias, ou seja, 2. Depois disso, é adicionado à data atual, estes 2 dias e é comparado se o resultado do dia (do mês) é igual ao especificado na publicação, se isto se verificar, é considerado como uma publicação válida.

- O último caso é a deteção de datas que não se façam acompanhar de uma expressão temporal, como o "amanhã" ou "segunda-feira", por exemplo. Assim, na eventualidade de existir uma data expressa no formato tipicamente utilizado, dd/mm ou dd/mm/aaaa, esta é capturada e, na eventualidade da data atual, face à data capturada, não superar os 3 dias de diferença, a publicação é considerada válida.

- Seguidamente, é realizada uma análise mais profunda ao conteúdo da publicação, de modo a recolher as informações relativas às viagens, como a cidade de partida e chegada, onde são primeiramente descartadas todas as publicações que contenham a palavra "cheio", ou similar. Esta decisão foi considerada já que, após um estudo sobre alguns grupos de partilha de boleias, muitos utilizadores, ao invés de eliminarem a publicação quando o número de lugares disponíveis se esgotava, apenas editavam a própria publicação acrescentando, por exemplo, "cheio" ou "lotado". Assim, devido a isto, todas as publicações que contenham este tipo de palavras são descartadas, bem como, as informações associadas à mesma. Este processo pode ser particularmente complexo e penoso, uma vez que cada utilizador é livre de formular uma frase em livre-arbítrio, havendo inúmeras maneiras possíveis de identificar a origem e destino. Perante isto, é complexo formular um algoritmo que consiga, de uma forma 100% eficaz, interpretar e recolher, assertivamente as cidades especificadas. Exemplos de possíveis frases que expressam a oferta de viagens, são:

Tabela 19 – Exemplos de publicações consideradas como válidas (oferta de boleias)

Porto – **Lisboa**, hoje às 15h
Coimbra –> **Aveiro**, amanhã pelas 11:30
Aveiro ➡️ **Braga**. 22h. 25/07. 10€
Faro para Aveiro, no dia 14/08 pelas 16h, com custo de 25€

O foco nesta iteração é, sobretudo, extrair assertivamente a cidade de partida e de chegada, independentemente da representação que esteja presente na publicação.

O método utilizado para proceder à análise do conteúdo de cada publicação, bem como, a deteção das datas fundamentadas no tópico anterior, é através de expressões regulares.

Uma expressão regular, pode ser entendida como uma sequência de caracteres que definem um padrão de pesquisa, de modo a permitir encontrar um ou mais caracteres de interesse, como por exemplo palavras, números, ou padrões de caracteres particularmente interessantes para o desenvolvedor, num dado documento ou texto.

Estas são escritas numa linguagem formal e interpretadas por um *Regex Engine* (processador de expressões regulares), que permite assim a identificação dos caracteres que correspondam com a expressão regular desenvolvida [115].

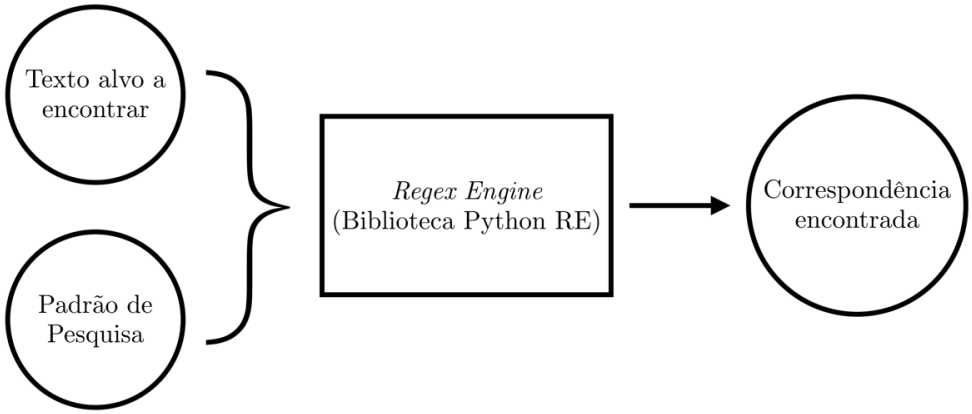


Figura 37 – Ciclo de etapas relativas às expressões regulares adaptado: https://goo.gl/UU3nGW

Assim, através deste mecanismo de procura, é possível formular expressões que captem as cidades de partida e de chegada, introduzidas livremente pelo utilizador aquando a publicação da partilha.

Tabela 20 – Estratégias de reconhecimento de cidades com expressões regulares

Separação pelo *emoji* da seta (➡️), podendo existir um ou mais espaços à esquerda e/ou à direita do *emoji*	Aveiro ➡️ Porto
Separação contendo obrigatoriamente, pelo menos um “>”, com a possibilidade de existir, pelo menos um “-“ e espaços	Aveiro > Porto Aveiro -> Porto Aveiro --> Porto
Separação contendo, obrigatoriamente, um “-“, podendo haver mais ocorrências deste e espaços	Aveiro – Porto Aveiro -- Porto Aveiro --- Porto
Separação contendo, obrigatoriamente, a palavra “para”	Aveiro para Porto

Desta forma, quando uma publicação obedece a uma das estratégias exemplificadas acima, é armazenado o resultado da expressão regular e procede-se a mais uma verificação.

Esta última verificação, irá avaliar a similaridade, tanto da cidade de origem como da de destino, com o conteúdo da lista de cidades de Portugal, previamente carregada em memória. A particularidade desta verificação, prende-se sobretudo com o facto do utilizador que submeteu a partilha da boleia, poder escrever erradamente uma destas cidades. Este ato é bastante comum em redes sociais e esta verificação visa corrigir possíveis erros, como por exemplo:

Tabela 21 – Correção ortográfica de cidades

Covilha em Covilhã
Evora em Évora
Braganca em Bragança

Se estas etapas se verificarem, a publicação é considerada como válida e, as informações associadas à mesma, serão armazenadas para posterior análise.
De notar que, na eventualidade de pelo menos uma das cidades, origem ou destino, não serem reconhecidas até ao final desta avaliação, a publicação é totalmente descartada.

- Concluída a etapa anterior da extração das cidades de origem e destino, sucede-se a avaliação da existência de um preço já estabelecido na publicação. Para este efeito, foi desenvolvida uma expressão regular, que capta as principais formas de representar o custo de uma viagem.

Tabela 22 – Estratégias de reconhecimento do custo com expressões regulares

Reconhecimento do valor da viagem, estando este **seguido ou precedido do sinal do euro**	5€ €5
Reconhecimento do valor da viagem, sendo este **composto por valores decimais, separados por um ponto final, estando seguido ou precedido do sinal do euro**	5.50€ €5.50
Reconhecimento do valor da viagem, sendo este **composto por valores decimais, separados por uma vírgula, estando seguido ou precedido do sinal do euro**	5,50€ €5,50
Reconhecimento do valor da viagem, sendo este **seguido da palavra euro/euros, com a possibilidade de existirem espaços**	5euro 5 euros
Reconhecimento do valor da viagem, sendo este **seguido da palavra Euro/Euros, com a possibilidade de existirem espaços**	5Euro 5 Euros
Reconhecimento do valor da viagem, sendo este **seguido da palavra EURO/EUROS, com a possibilidade de existirem espaços**	5EURO 5 EUROS

Ao verificar-se assim uma das ocorrências exemplo, demonstradas acima, a publicação é considerada como válida, associando-se um preço à mesma. No decorrer deste processo, o resultado da expressão regular, passa por uma função de limpeza, onde, na eventualidade desta não ter conseguido apenas captar o preço, são eliminados todos os caracteres que não sejam dígitos, pontos finais, vírgulas ou o símbolo do euro.

Na hipótese de o preço não estar definido na publicação, ao contrário do que foi apresentado nas verificações anteriores, esta não é descartada, uma vez que existe a possibilidade do preço ser acordado pessoalmente.

- A última verificação considerada antes de se proceder à construção da resposta em JSON, é a deteção da hora de partida da viagem partilhada. De notar que, apenas as publicações que se façam acompanhar de hora de partida, é que serão consideradas como válidas, uma vez que, à semelhança da cidade de partida e de chegada, bem como a data, são fatores indispensáveis para o agendamento de uma viagem e para a construção da resposta JSON.

  Assim, de modo a verificar-se a existência de uma hora presente na publicação, foram desenvolvidas e consideradas quatro expressões regulares, que captam as formas mais usuais de representar a hora de uma viagem.

Desta forma, quando uma publicação obedece a uma das estratégias apresentadas abaixo, a publicação e a informação associada à mesma são armazenadas, caso contrário, todas as informações são descartadas.

Tabela 23 – Estratégias de reconhecimento da hora de partida com expressões regulares

Reconhecimento da hora e dos minutos de partida da viagem estando estes separadas pelo delimitador ":"	10:30
Reconhecimento da hora e dos minutos de partida da viagem estando estes separadas pelo caracter H, podendo este ser minúsculo ou maiúsculo	10h30 10H30
Reconhecimento da hora de partida da viagem estando esta seguida da palavra "horas", podendo apresentar um espaço	10horas 10 horas
Reconhecimento da hora de partida da viagem estando esta seguida do caracter "H", podendo este ser minúsculo ou maiúsculo	10h 10H

Como conclusão, o fluxograma que se segue, pretende sumariar as principais etapas, previamente detalhadas, da solução empregue a cada publicação recolhida através de um grupo da rede social Facebook.

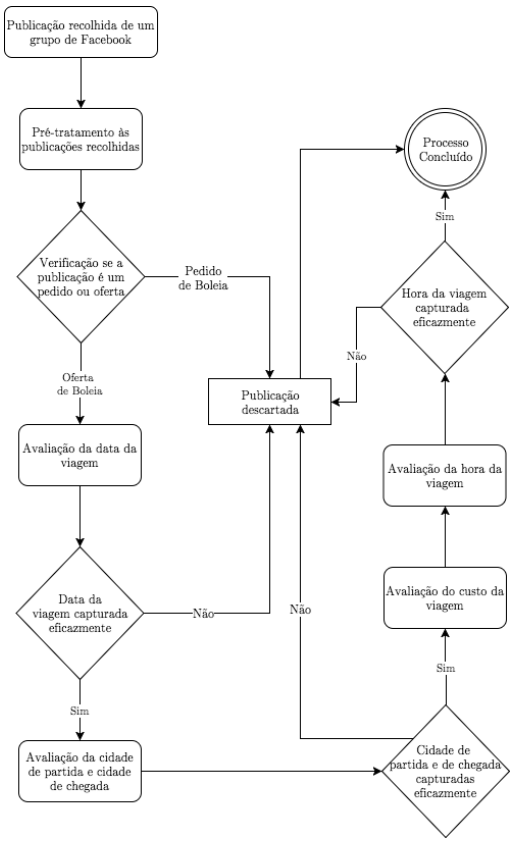


Figura 38 – Fluxograma relativo às técnicas de Text Mining aplicadas a cada publicação

Relativamente à execução automática do *crawler*, este não pode ser realizada sem serem aplicadas verificações relativas ao conteúdo das publicações de cada grupo, de modo a evitar a população da base de dados com publicações repetidas.

Desta forma, a cada hora o *crawler* é iniciado e é estabelecida conexão com cada um dos grupos do Facebook, cujo ID tenha sido armazenado pelo administrador do sistema, através da interface de administração da solução.

Uma vez que a execução do *crawler* é periódica, pode tanto acontecer que na nova iteração deste, existam novas publicações, como não. Em ambas as eventualidades, é necessário condicionar a execução do *crawler*, para que entradas redundantes não sejam adicionadas à base de dados.

Assim, em cada iniciação do *crawler*, é tomado em consideração o último registo na base de dados relativo ao grupo onde o *crawler* está a ser realizado e comparado com a primeira publicação na *stack* de publicações recolhidas, uma vez que a ordem em que os resultados são organizados, é a mesma dos que são apresentados na *timeline* do grupo. Se se verificar a igualdade destes, significa que não foram inseridas novas publicações no determinado grupo e que não se irá proceder ao processamento dos dados.

Por outro lado, ao consultar a última entrada na BD de um determinado grupo e se este não retornar nenhum resultado, significa que ainda nenhum *crawler* foi realizado para o determinado grupo e assim, todo fluxo do tratamento de dados, bem como a adição à base de dados, será executada.

Por último, ou seja, se já houverem registos na base de dados relativos a um grupo, é iterada a lista relativa à data/hora das publicações recolhidas e comparada com a última data/hora da última publicação desse mesmo grupo. Deste modo, quando a i-ésima posição da lista for igual ao último registo da base de dados, apenas serão consideradas as publicações recolhidas até à posição anterior à i-ésima publicação recolhida, ou seja, as novas publicações até ser alcançada a publicação já armazenada.

O fluxograma seguinte, sumaria as etapas anteriormente especificadas anteriormente, sustentando o raciocínio empregue.

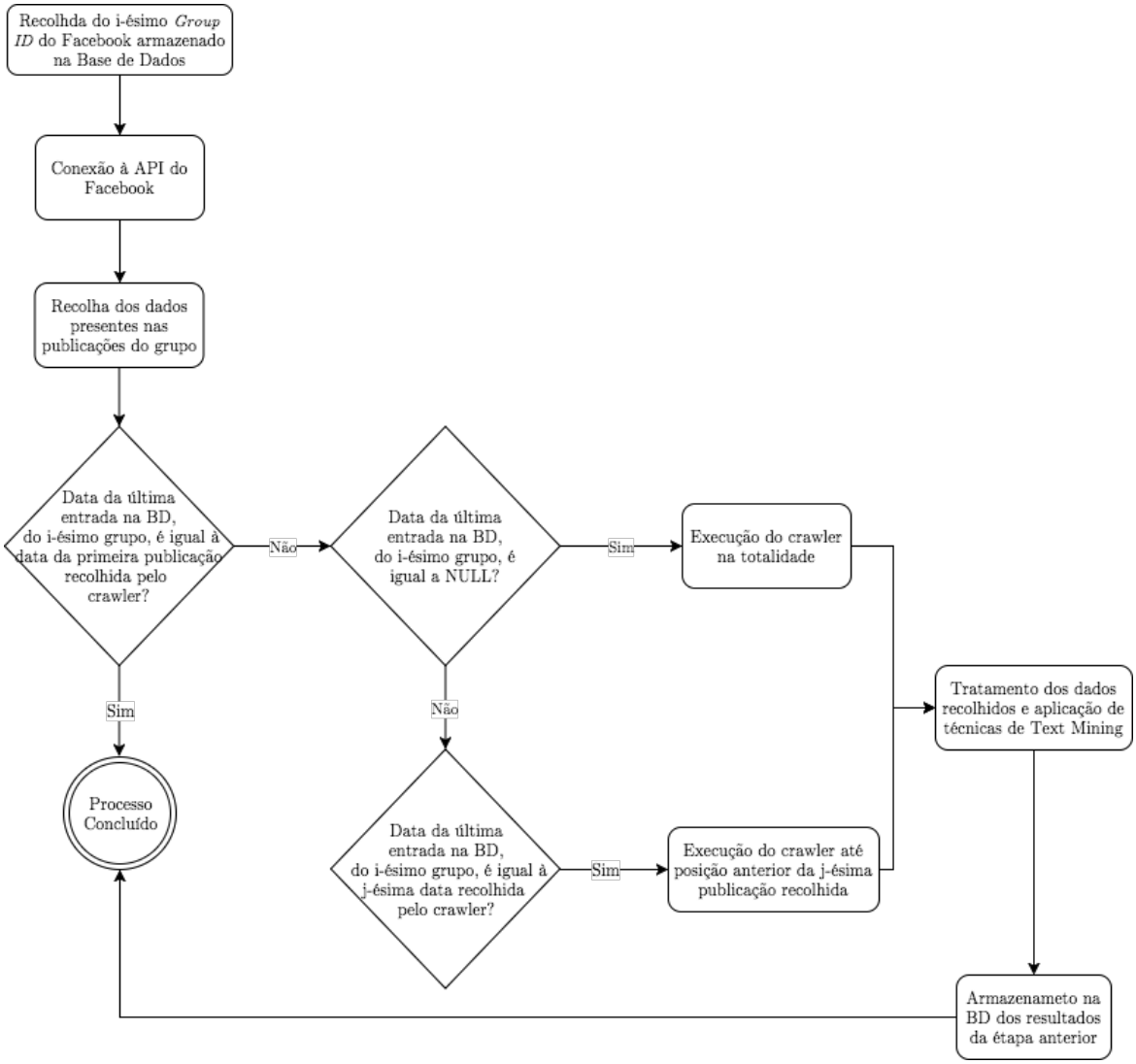


Figura 39 – Fluxograma relativo à gestão de novas publicações pelo crawler

4.5. Representação dos Recursos

Para representar os resultados de uma consulta, recorreu-se à notação JSON, onde foram consideradas algumas das boas práticas na construção das representações na notação JSON. Parte destas boas práticas, são baseadas nas recomendações elaboradas pela Google [116].

Primeiramente, é importante lembrar que o JSON é uma notação assente em duas estruturas: coleção de pares chave-valor, frequentemente denominada por objeto e, lista ordenada de valores.

Uma chave, é uma *string* rodeada por aspas, sendo que um valor pode ser uma *string*, um número, um valor booleano (*True* ou *False*), uma lista, um objeto ou um valor nulo (*Null*).

A Listagem 2, apresenta a estrutura de um objeto JSON, composto por um par chave-valor. Na eventualidade em que o objeto é composto por mais do que um destes, os pares são separados por uma vírgula.

Listagem 2 – Objeto JSON

```
{
  “chave”: valor
}
```

Em relação à lista ordenada de valores, a sua estrutura em JSON pode ser visualizada na Listagem 3 e, tal como no caso dos objetos JSON, se a lista for composta por mais que um objeto, então os valores são separados por vírgula.

Listagem 3 - Lista ordenada de valores em JSON

```
[
  valor1,
  valor2
]
```

4.5.1. Resposta JSON

A definição e construção da resposta que é apresentada ao utilizador, após este efetuar uma solicitação à API, é sem dúvida uma das partes mais significativas e imprescindíveis na realização desta solução.

A estruturação dos elementos da resposta, tenciona ser percetível tanto na leitura por parte do *developer* que a consulta, bem como apresentar uma arquitetura simples para a extração de dados a partir da mesma.

A resposta pode ser assim segmentada em dois fragmentos, como ilustrado na figura seguinte – Figura 40.

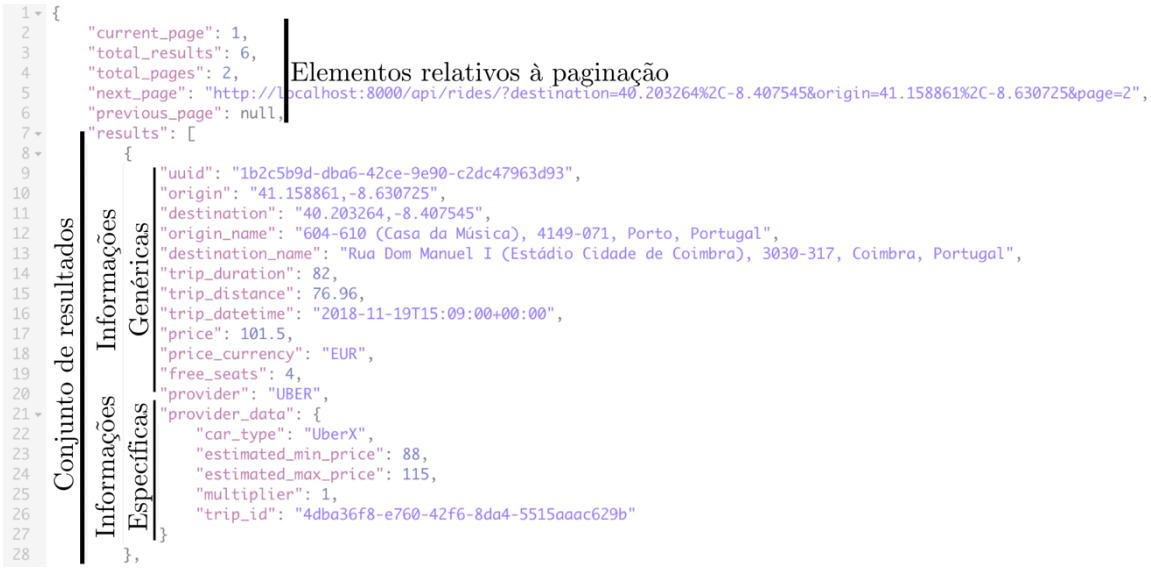


Figura 40 – Composição de uma resposta JSON

Como observável, os **elementos relativos à paginação** são compostos por três campos, sendo esses:

- *current_page*: página atual da resposta
- *total_results:* número de elementos do conjunto de resultados, isto é, o número de viagens disponíveis
- *total_pages:* número de páginas que a resposta apresenta
- *next_page*: hiperligação para a página seguinte de resultados
- *previous_page*: hiperligação para a página anterior de resultados

Do mesmo modo, como observável, o **conjunto de resultados** é uma lista de valores, que agrega as informações e estatísticas dos *provider*s que ofereçam viagens para as coordenadas especificadas no pedido. Neste, a informação apresentada está segmentada em duas parcelas, as informações genéricas a qualquer *provider* e as que apenas dizem respeito a um dado *provider*.

> As *informações genéricas* podem ser caracterizadas como informações que se mantêm em comum independentemente do *provider* que apresente viagens, ou seja, informações imprescindíveis para quem irá usufruir do serviço, como por exemplo, a origem, o destino e o preço.

Por outro lado, as *informações específicas* reúnem informações que irão oscilar consoante o *provider* que disponha de viagens para as coordenadas especificadas no pedido.

As tabelas abaixo, apresentam assim as informações específicas relativas a cada *provider*.

Tabela 24 – Informações específicas da UBER contidas na resposta JSON

provider_data	car_type	Designação do automóvel atribuído para a viagem
	low_estimate	Estimativa do preço mínimo aplicado à viagem
	high_estimate	Estimativa do preço máximo aplicado à viagem
	multiplier	Multiplicador de preço aplicado na viagem
	trip_id	Identificador da viagem na plataforma do Uber

Tabela 25 – Informações específicas da LYFT contidas na resposta JSON

provider_data	estimated_min_price	Estimativa do preço mínimo aplicado à viagem
	estimated_max_price	Estimativa do preço máximo aplicado à viagem
	trip_id	Identificador da viagem na plataforma do Lyft

Tabela 26 – Informações específicas da TaxiFareFinder contidas na resposta JSON

provider_data	price_without_commissions	Preço da viagem sem comissõ
	initial_fare	Tarifa inicial aplicada à viagem
	tip_amount	Gorjeta aplicada à viagem, não aplicável em todos os países
	extra_charge	Comissão extra aplicada
	extra_charge_description	Descrição da comissão extra aplicada
	available_taxi_info	Informações relativas à companhia taxista, como, nome e número de telefone

Tabela 27 – Informações específicas da BlaBlaCar contidas na resposta JSON

provider_data	car_type	id	Identificador do veículo na plataforma do BlaBlaCar
		make	Marca do veículo atribuído para a viagem
		model	Modelo do veículo atribuído para a viagem
		confort	Designação de conforto conferido ao automóvel atribuído para a viagem
		confort_nb_star	Grau de conforto conferido ao automóvel atribuído para a viagem
	trip_id	Identificador da viagem na plataforma do BlaBlaCar	

Tabela 28 – Informações específicas da Facebook contidas na resposta JSON

provider_data	post_msg	Corpo da publicação
	created_time	Data de publicação
	facebook_publisher_name	Nome do utilizador na plataforma do Facebook
	facebook_publisher_id	Identificador do utilizador na plataforma do Facebook
	facebook_group_id	Identificador do grupo na plataforma do Facebook

4.5.2. Serialização dos Dados

O conceito de serialização dos dados, pode ser formalmente expresso como um processo de tradução de estruturas de dados ou do estado de um objeto, num formato que possa ser armazenado (como por exemplo, um ficheiro ou *buffer* de memória) ou transmitido (como por exemplo, através de uma conexão à internet), onde é posteriormente reconstruído, possível de ser no mesmo ambiente computacional ou num distinto [117].

Quando aplicado ao desenvolvimento de APIs REST, permite a recolha dos objetos contidos nos *Models*, para que seja possível, posteriormente, o consumo em serviços *Web*. No entanto, aquando da recolha dos dados, é possível especificar os campos que se deseja retornar, de modo a evitar o envio de dados que sejam confidenciais, como é o caso de *passwords* ou de *tokens* de acesso.

O formato de retorno dos dados, após a serialização é em JSON, uma forma textual de representação de dados estruturados numa coleção de pares chave-valor, onde a chave é expressa em texto e o valor pode ser tanto representado em texto, números (inteiros ou decimais), valores booleanos, valores nulos, como numa sequência ordenada de valores.

Este formato é adequado para o contexto de APIs REST, uma vez que é independente da linguagem de programação, de fácil criação, manipulação e de análise [118].

4.5.3. Paginação

O conceito de paginação está presente num vasto número de *sites*, como *blogs*, *sites* de notícias e até mesmo em motores de pesquisa. Normalmente, a paginação é utilizada quando o conteúdo a representar, excede o espaço disponível na página, devido ao número de elementos presentes na resposta a apresentar, ou até mesmo para segmentar os resultados a apresentar ao utilizador.

A representação gráfica mais usual, associada a este conceito, e que permite a navegação entre páginas, é através da enumeração das mesmas.

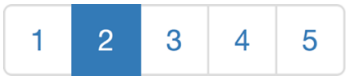


Figura 41 – Representação Gráfica da Paginação aplicada a Interfaces Web adaptado: https://goo.gl/HRUY4P

Contudo, quando aplicada ao desenvolvimento de APIs REST, a paginação opera de um modo diferente, porém, com o mesmo conceito subjacente. E, ao recuperar um resultado de um pedido, onde o conjunto de dados seja vasto, é possível que se possam enfrentar algumas dificuldades.

Sabendo que a API possui vários métodos que retornam uma coleção de dados por pedido, é possível que a coleção de dados a retornar seja extensa. Com isto, é necessária uma grande capacidade, por parte dos clientes, para a processar a mesma, uma vez que os resultados serão descarregados e processados a cada pedido realizado a um dado *endpoint*, o que não se torna flexível devido aos custos da rede associados à operação. O ênfase desta preocupação, prende-se sobretudo pelo facto de, atualmente, grande parte dos acessos destes serviços serem realizados em dispositivos móveis. Associada a essa grande quantidade de dados, pode estar também a possibilidade de elementos desnecessários que requereram, de igual forma, de poder computacional.

Com vista a solucionar este problema, existem duas abordagens possíveis de implementar:

A primeira abordagem, é implementar a paginação do "lado do servidor". Esta abordagem é adequada e indicada, quando a quantidade de dados a paginar é elevada, ou quando se prevê que venha a ser bastante elevada, tendo como principal vantagem uma apresentação rápida e segmentada da informação ao cliente. São estes, geralmente, aplicações móveis ou páginas *Web*.

A segunda abordagem consiste no envio completo de coleções para os clientes e estes, posteriormente, aplicam a paginação, apresentando aos utilizadores, um conjunto de *itens* de cada vez. Esta abordagem é indicada quando a quantidade a paginar é reduzida.

Deste modo, e devido à possível quantidade de dados – entenda-se viagens disponíveis para as coordenadas inseridas – que certas coleções podem conter, o tipo de paginação adotado na API desenvolvida é a abordagem do "lado do servidor". A solução passa por segmentar o resultado proveniente do pedido realizado em páginas, onde é tido em consideração o tamanho total da coleção e o limite de itens a paginar, diminuindo assim o tamanho de cada resposta e consequentemente os custos de rede, uma vez que apenas uma parcela da resposta, será processada e devolvida.

Tomando como exemplo uma resposta que retorne uma coleção de N objetos JSON, é possível estabelecer um número limite de objetos JSON que estejam presentes em cada página.

Assim, se a resposta for composta por 21 objetos JSON e estabelecendo um número máximo de 5 objetos por página, teremos 5 páginas de resultados, onde só serão processados os 5 seguintes objetos quando for realizado um novo pedido, especificando o número da página desejada. Desta forma, a API tem implementado, de modo estandardizado, o retorno de até 5 itens por cada pedido, a qualquer um dos métodos que devolva uma coleção de dados. Porém, e na eventualidade do conjunto retornado exceder 5 itens, são fornecidas na resposta ligações, de modo a permitir a navegação para os restantes itens da coleção.

Outra funcionalidade implementada e disponível na utilização da API, é a especificação do número de elementos a paginar através do URL do pedido. Ao utilizar este recurso, o valor padrão de 5 elementos por página é *overrided*, tomando assim o valor inserido pelo utilizador no URL, podendo este tomar valores menores ou maiores do valor previamente estabelecido, como evidenciado nas Figuras 42 e 43. [119] [120] [121]

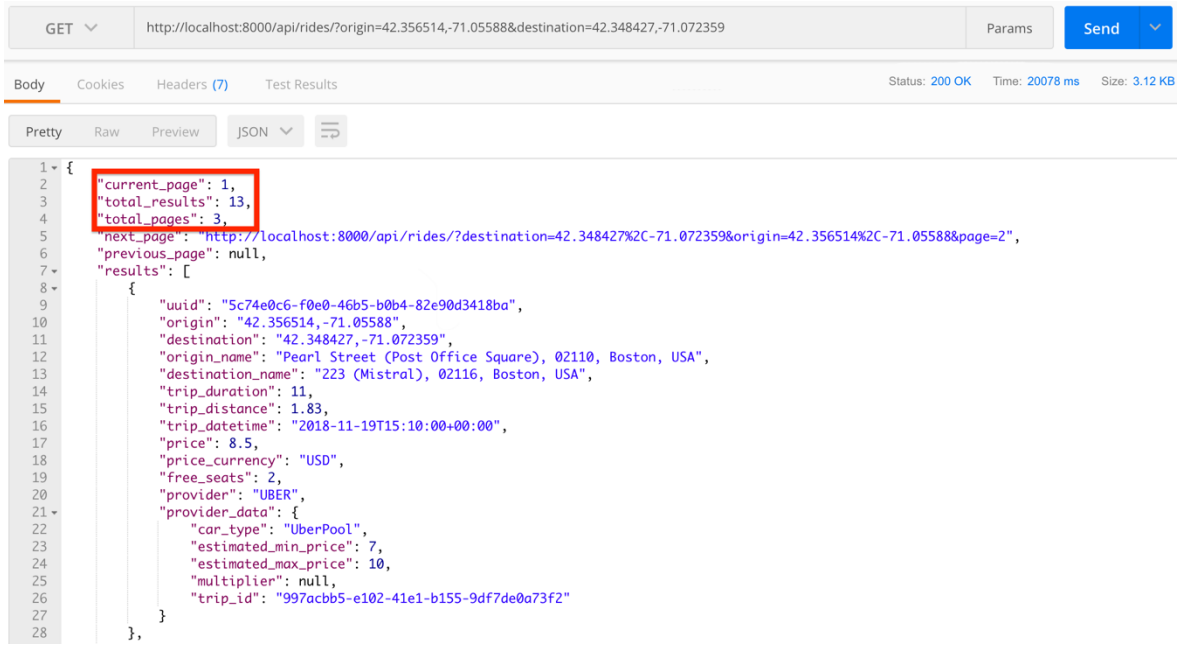


Figura 42 – Resultado de uma pesquisa com paginação padrão de 5 elementos

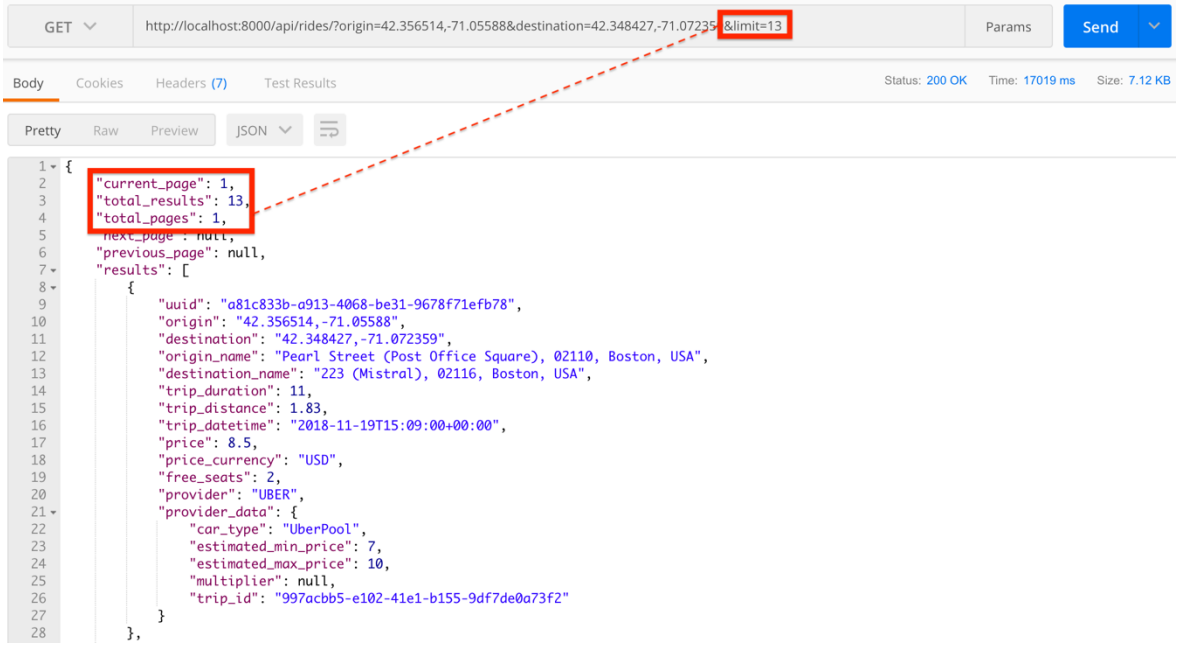


Figura 43 – Resultado de uma pesquisa com a paginação definida pelo URL de pesquisa

É pertinente também mencionar a possibilidade da coleção ser alterada, tanto pela adição de novos dados – como é o exemplo de haver um pedido realizado instantes após o anterior – como na remoção de dados já existentes – como por exemplo na manutenção da base de dados por parte de um *superuser* –, onde o esquema de paginação, baseado em páginas, pode levar a que o servidor envie dados repetidos em diferentes páginas, como ilustrado na Figura 45.

A Figura 44 demonstra, por outro lado, o correto o funcionamento do esquema baseado em páginas.

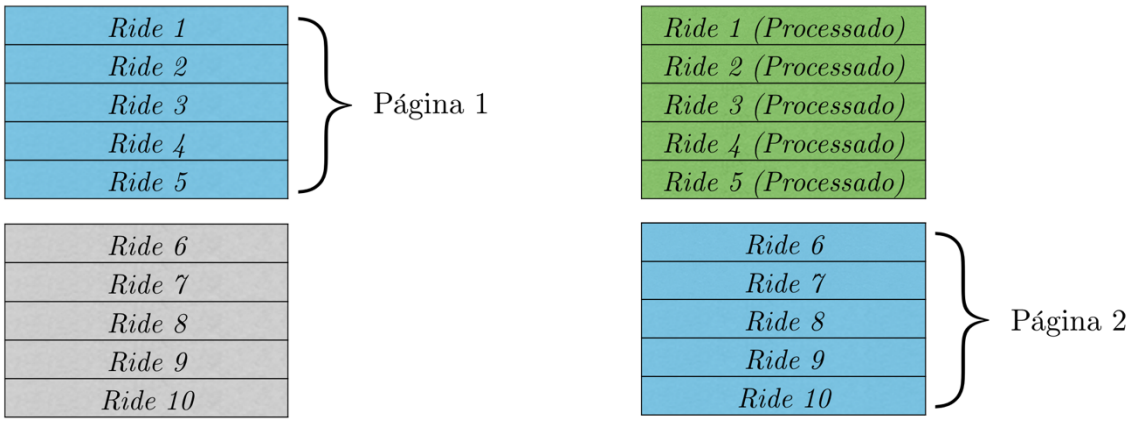


Figura 44 – Paginação usando um esquema baseado em páginas

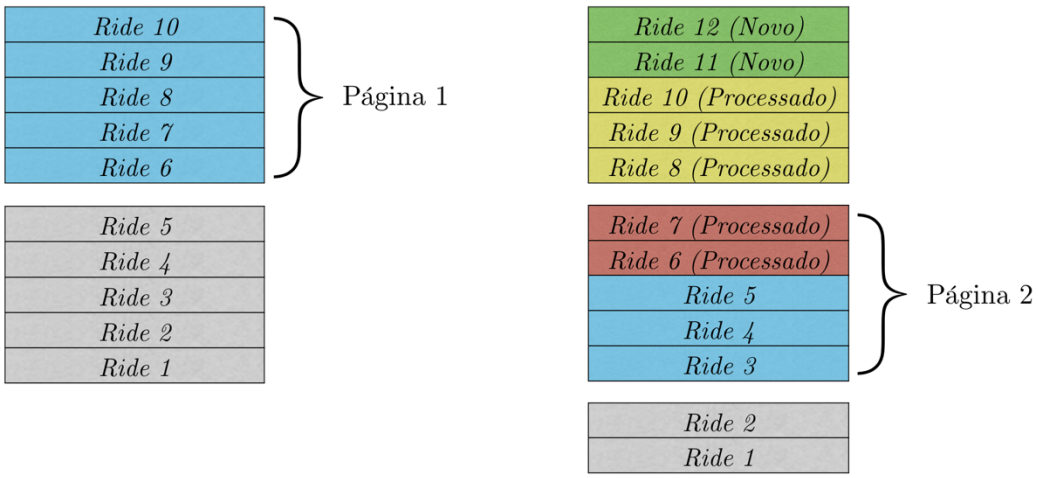


Figura 45 – Possível problema do esquema baseado em páginas

Neste exemplo, os dados estão organizados do mais recente para o mais antigo e, entre o primeiro e o segundo pedido do cliente, surgem dois novos *itens*. Como só é tomado em consideração o número de itens em cada página, existirão dois itens que se repetem na segunda resposta da API. Assim, e de modo a solucionar este problema, onde existe a possibilidade de a base de dados ser alterada, é possível adotar a paginação com esquema baseado em cursores, ao invés do esquema baseado em páginas. Neste, os itens são retornados de acordo com a posição de um cursor na coleção. Deste modo, quando um cliente realiza um pedido ao servidor, este envia o conjunto de itens e a posição em que o cursor se encontra, que será o item seguinte ao primeiro conjunto. No pedido seguinte, o cliente pode usar o cursor para indicar que quer o segundo conjunto de itens a começar na posição do cursor. Na Figura 46, é apresentado o funcionamento do esquema baseado em cursores e a maneira como é evitado o problema inerente ao esquema baseado em páginas. À vista disso, um cliente pode navegar pela coleção utilizando os dois parâmetros, *before* e *after*, no URL de pesquisa, tendo que fornecer um identificador de um cursor como valor do parâmetro. Ao colocar então o parâmetro *after* e um identificador de um cursor no URL, a API retorna os itens subsequentes à posição do cursor. De igual modo, o parâmetro *before*, tem um funcionamento igual, com a diferença de retornar os itens anteriores à posição do cursor. De notar que, o esquema baseado em cursores, apenas apresenta os controlos de avanço e retrocesso, não permitindo assim que o cliente navegue para posições arbitrárias, ao contrário do baseado em páginas. [122]

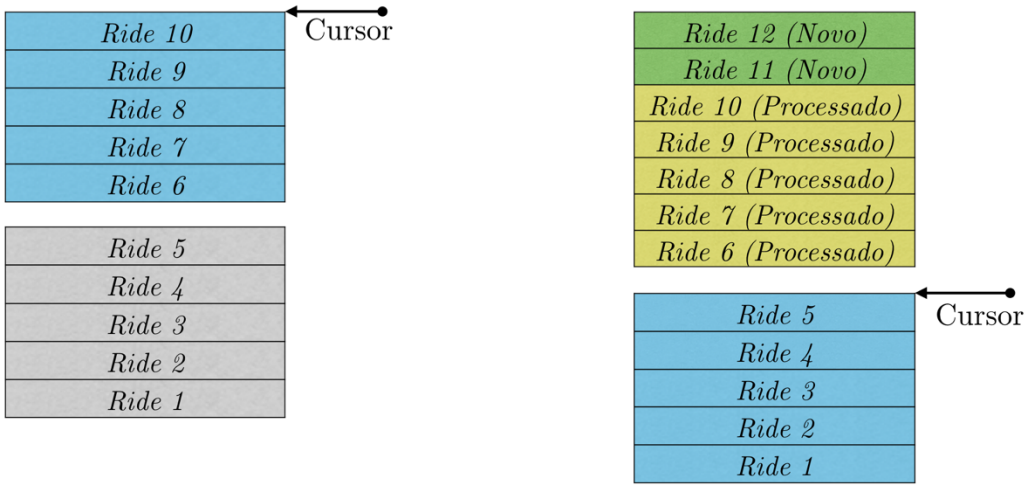


Figura 46 – Paginação usando um esquema baseado em cursores

Contudo, na solução implementada, não houve necessidade de adotar o esquema baseado em cursores, elegendo assim o esquema baseado em páginas. Uma vez que a base de dados é apenas alimentada aquando da solicitação de um método GET da API, onde o processo inerente é a conexão às respetivas APIs dos *providers* disponíveis, a recolha e o armazenamento na base de dados e posterior retorno da informação processada ao utilizador. Sabendo que a transição de um estado para o seguinte, pressupõe a completa realização dos mesmos, a possível adversidade exposta anteriormente não acontecerá, uma vez que a escrita para a base de dados só ocorre durante a solicitação de um pedido. Outro fator que também reduz a ocorrência de serem retornados valores indesejados, é a adição do *timestamp* da pesquisa aquando da recolha e serialização dos dados presentes da base de dados.

No que diz respeito à organização dos elementos relativos à paginação na resposta apresentada após a elaboração de um pedido, esta pode ser livremente redefinida pelo desenvolvedor, de modo a acrescentar mais informações que enriqueçam e facilitem a integração da REST API em outras aplicações.

Desta forma, a Listagem 4, apresenta os elementos predefinidos aquando da integração do módulo *Pagination*, presente na biblioteca Django Rest Framework, e a Listagem 5 os elementos redefinidos que acompanham a solução final.

Listagem 4 – Elementos preestabelecidos na Paginação do DRF

```
{
  “count”: Número total de elementos,
  “next”: Hiperligação para página seguinte de resultados,
  “previous”: Hiperligação para página anterior de resultados,
  “results”: [
      Conjunto de resultados
  ]
}
```

Listagem 5 – Elementos presentes na solução final

```
{
  “current_page”: Página atual da resposta,
  “total_results”: Número total de elementos,
  “total_pages”: Número total de páginas,
  “next_page”: Hiperligação para página seguinte de resultados,
  “previous_page”: Hiperligação para página anterior de resultados,
  “results”: [
      Conjunto de resultados
  ]
}
```

4.6. Proposta de Arquitetura

Todos os conceitos apresentados até ao presente ponto, contribuíram, direta e indiretamente, para apresentar a arquitetura da solução que se segue. A Figura 47, sumariza assim os principais módulos que constituem o funcionamento da solução. De um modo sucinto, o processo pode ser resumido nas seguintes etapas:

- O cliente realiza um pedido HTTP ao ASMA API, onde para além da introdução dos parâmetros obrigatórios e/ou opcionais a um dado *endpoint*, esteja incluído um *access-token* no cabeçalho do pedido, de modo a verificar a legitimidade do mesmo.
- Seguidamente, o pedido é processado e, dependendo do *endpoint* introduzido no pedido, é consultado um ou mais *providers* disponíveis na solução, onde serão recolhidas as informações relativas às viagens oferecidas. Concluída esta etapa, onde se verificou a recolha dos dados, os mesmos são armazenados na base de dados.

- Paralelamente, e em intervalos de 1 em 1 hora, é realizado um *crawler* a todos os grupos de Facebook dedicados à partilha de viagens, armazenados na solução, onde são recolhidas as publicações relativas à oferta de viagens. Essas publicações são posteriormente submetidas a um processo de análise e processamento de linguagem natural que, ao verificar um conjunto de verificações, ditará a relevância da mesma e serão assim recolhidas as informações presentes. O processo termina com o registo na base de dados das informações resultantes do tratamento aplicado.
- A última etapa consiste em devolver ao cliente uma resposta HTTP, onde esteja presente toda a informação recolhida e desejável, mediante a consulta realizada.

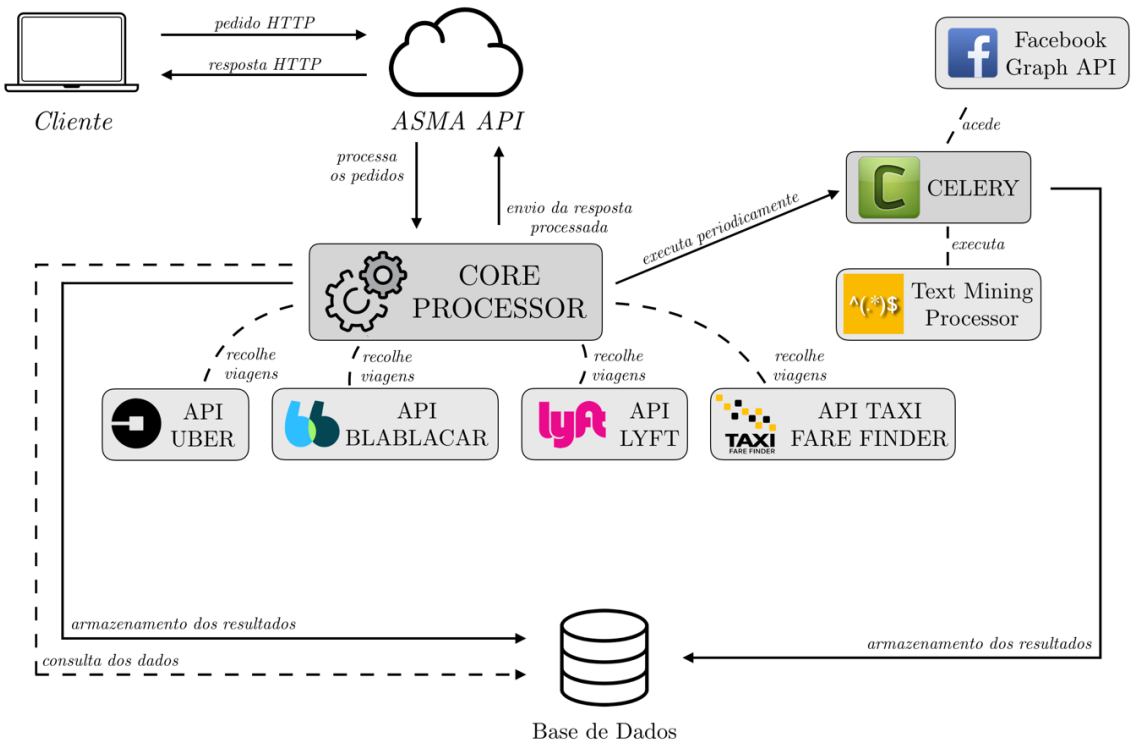


Figura 47 – Arquitetura da solução

4.7. Bibliotecas Utilizadas

No desenvolvimento do projeto foram utilizadas várias bibliotecas que complementaram e deram suporte ao desenvolvimento da solução, descrita nos tópicos anteriores.

As mesmas são apresentadas de seguida:

- **Requests**[50] – Biblioteca de auxílio ao desenvolvedor na execução de pedidos HTTP a serviços *Web*.

- **GeopositionField**[51] – Tipo de atributo que permite o armazenamento de uma geo-posição (latitude e longitude), nos *Models*.

- **Uber Rides**[52] – SDK oficial, de apoio à utilização da API Uber Rides. Para uma correta utilização, é necessário o registo prévio nos serviços Uber e a criação de uma nova aplicação, com o propósito de obter um *server-token*, de modo a que seja possível estabelecer ligação e obter informações relativas às viagens.

- **Lyft Rides**[53] – SDK não oficial, de apoio à utilização da API Lyft Rides. Esta opera de modo idêntico ao da Uber, onde é necessário também um registo e criação de uma aplicação, para obter as credenciais (*cliente_id* e *client_secret*) e para que seja possível a posterior conexão e obtenção da informação relativa às viagens.

- **Google Maps API**[54] – Biblioteca em Python, de conexão à API Google Maps. Para a utilização desta, é necessária a aquisição de um access-token, possível de ser obtido gratuitamente https://developers.google.com/console com uma conta Google. Com esta biblioteca é possível a conexão à API de direções, matriz de distância, geo-codificação, geo-localização, entre outras...

[50] https://pypi.org/project/requests
[51] https://pypi.org/project/django-geoposition
[52] https://pypi.org/project/uber_rides
[53] https://pypi.org/project/lyft_rides
[54] https://pypi.org/project/googlemaps

- **GeoCoder**[55] – Biblioteca escrita em Python, de geo-codificação. Vários provedores de serviços de geo-codificação, como é o caso da Google e do Bing, não incluem bibliotecas Python e a resposta retornada é diferente de provedor para provedor.
  Assim, de modo a evitar a elaboração manual de pedidos HTTP e posterior *parse* a cada *provider* de geo-codificação, é utilizada esta biblioteca. Neste momento, a biblioteca GeoCoder conta com mais de 25 *providers* possíveis de utilizar e é capaz de prover várias informações nas suas respostas, relativas a uma dada coordenada, cidade, país, morada e ainda a obtenção das coordenadas atuais de um dispositivo.

```
>>> import requests
>>> url = 'https://maps.googleapis.com/maps/api/geocode/json'
>>> params = {'sensor': 'false', 'address': 'Mountain View, CA'}
>>> r = requests.get(url, params=params)
>>> results = r.json()['results']
>>> location = results[0]['geometry']['location']
>>> location['lat'], location['lng']
(37.3860517, -122.0838511)
```

Figura 48 – Exemplo de um pedido HTTP e parse aos serviços Google

```
>>> import geocoder
>>> g = geocoder.google('Mountain View, CA
>>> g.latlng
(37.3860517, -122.0838511)
```

Figura 49 – Exemplo de um pedido aos serviços Google através da biblioteca GeoCoder

- **GeoPy**[56] – Biblioteca que aglomera vários *Web-Services* relativos a geo-codificação, onde é possível apontar o OpenStreetMap Nomination[57], uma ferramenta *open-source* que oferece vários serviços, alguns apenas disponíveis em ferramentas pagas, de forma gratuita. O GeoPy permite, de forma fácil para os desenvolvedores Python, localizar as coordenadas relativas a moradas, cidades, países de modo semelhante ao GeoCoder, com a particularidade de permite também o cálculo da distância entre dois pares de coordenadas.

[55] https://pypi.org/project/geocoder

[56] https://pypi.org/project/geopy

[57] https://wiki.openstreetmap.org/wiki/Nominatim

- **Arrow**[58] – Biblioteca Python, que oferece uma abordagem mais simplista e amigável para criar, manipular, formatar e converter datas, horas e *timestamps*. Com esta biblioteca, é possível manipular datas e horas recorrendo a menos *imports* e utilizando menos código.

- **Django Preferences**[59] – Módulo destinado ao Django que permite aos utilizadores definirem preferências específicas da aplicação desenvolvida através da interface de administrador. Através desta, torna-se útil e fácil o armazenamento de *tokens* de acessos dos *providers*. Assim, cada entrada terá informação relativa a cada provider, como *client_id*, *client_secret* e *token*.

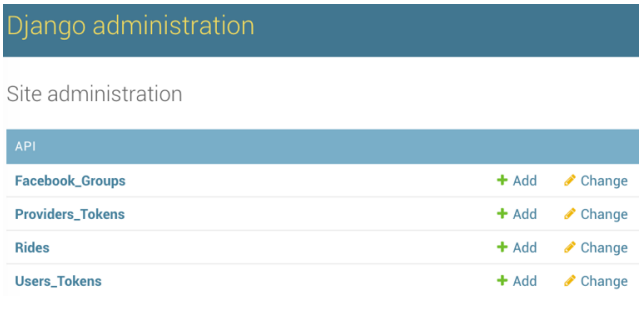


Figura 50 – Excerto da interface de administrador

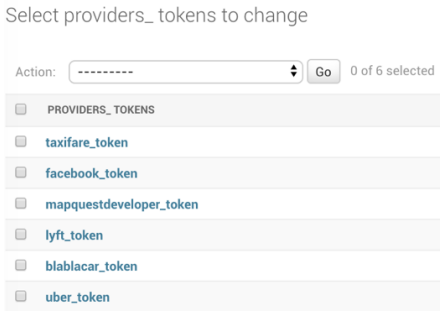


Figura 51 – Entradas do Providers_ Tokens

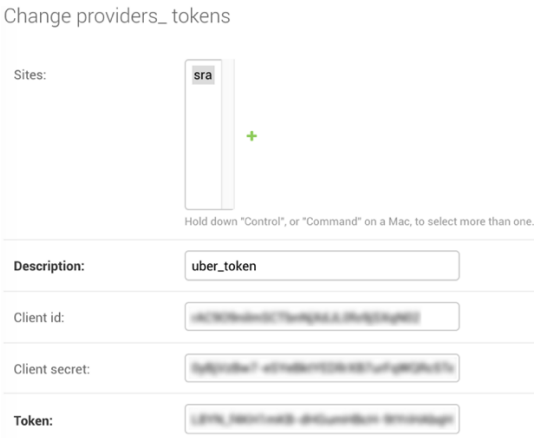


Figura 52 – Providers_ Tokens (Uber)

[58] https://pypi.org/project/arrow

[59] https://pypi.org/project/django-preference

- **NLTK**[60] – O *Natural Language Toolkit*, ou apenas NLTK, é um conjunto de bibliotecas que permitem efetuar o *Natual Language Processing*. O principal foco desta biblioteca é em aplicações de Inteligência Artificial, apresentando funcionalidades que permitem tratar interações, em texto, entre humanos e máquinas. Exemplos da interação com esta biblioteca, é a possibilidade de classificação, tokenização e stemização.

- **Swagger**[61] – Gerador de documentação semi-automático de APIs para a framework Django REST Framework, com uma interface gráfica, onde é possível consultar os detalhes da API, fornecida pelo próprio Swagger.

- **JSON**[62] – Módulo que possibilita a criação de objetos JSON. É possível salientar os seguintes métodos, úteis na serialização e desserialização dos objetos JSON contidos nas respostas provenientes dos *providers:*
  - `dumps()` – Permite a serialização de um objeto Python numa *string* JSON.

  - `loads()` – Permite a desserialização de um objeto *string*, *bytes* ou *bytearray*, que contenha um documento JSON, num objeto Python.

- **UUID**[63] – Módulo que permite a criação de objetos UUID – *Universally Unique IDentifier* – imutáveis, o que se torna praticamente útil, uma vez que, cada viagem armazenada na base de dados, terá de ter uma identificação associada.

[60] http://www.nltk.org

[61] https://pypi.org/project/django-rest-swagger

[62] https://docs.python.org/3.6/library/json.html

[63] https://docs.python.org/3.6/library/uuid.html

Este módulo fornece quatro métodos distintos, que permitem a criação de um UUID, que são:

- `uuid1()` – Gera um UUID com base no ID do computador e no horário atual. Este método não é o mais recomendável, uma vez que pode comprometer a privacidade, já que é utilizado o endereço de rede do computador.

- `uuid3()` – Gera um UUID, utilizando uma *hash* MD5 de um identificador *namespace* (que é um UUID já existente) e um nome (*string* de texto). Este método não foi escolhido, uma vez que consome mais poder computacional face aos outros métodos existentes e por não se ágil na construção de vários UUID.

- `uuid4()` – Gera um UUID de forma aleatória. Este foi o método escolhido para a geração de UUIDs já que utiliza menos poder computacional face aos outros métodos, sendo a probabilidade de gerar dois UUID iguais é infinitamente pequena.

- `uuid5()` – Gera um UUID baseado numa *hash* SHA-1 de um identificador *namespace* e num nome, de igual modo que o método uuid3, apresentando a mesma desvantagem.

- **Datetime**[64] – O módulo Datetime fornece classes e métodos que permitem a manipulação de datas e horas, de forma simples e complexa. É possível recolher a data e hora atual, e aplicar aritmética tanto de datas como de horas, através da adição ou subtração de segundos, minutos, horas, dias, entre outras. Deste modo, e com o auxílio da biblioteca arrow é possível a criação de um *timestamp*, onde está contida a data, hora e a diferença do fuso horário face ao Meridiano de Greenwich.

[64] https://docs.python.org/3.6/library/datetime.html

- **Re**[65] – Módulo que permite a realização de operações de correspondência de expressões regulares.

- **Dfflib**[66] – Módulo que permite comparar sequências entre objetos. A integração deste na solução, foi com o propósito de avaliar a similaridade entre palavras, retornando a mais exata, quando comparada com uma lista de *strings*.

4.8. Metodologias e Técnicas de Desenvolvimento

A elaboração da componente prática da presente dissertação teve como base uma metodologia de investigação orientada à engenharia [123]. Esta metodologia, assenta, normalmente, num conjunto de 7 passos que visam aos desenvolvedores de sistemas de informação a planearem a arquitetura do seu sistema e a encontrarem mais facilmente respostas aos desafios lançados. As etapas são apresentadas de seguida, contribuindo para um desenvolvimento iterativo e incremental da solução [123]:

- Definição do problema;
- Investigação da temática;
- Especificação dos requisitos;
- Análise e discussão de soluções/*frameworks* para o desenvolvimento do desafio lançado;
- Desenvolvimento da solução;
- Criação de protótipos;
- Elaboração de testes à solução e possível *refactor*.

Para o desenvolvimento do desafio proposto foram consideradas várias etapas do ciclo de vida do desenvolvimento de um *software* – *Systems Development Life Cycle* (SDLC) – definidos por Saini e Kaur em *A Review of Open Source Software Development Life Cycle Models* [124]. Este é composto por quatro fases principais: conceção, projeção, criação e implementação.

---

[65] https://docs.python.org/3.6/library/re.html

[66] https://docs.python.org/3.6/library/difflib.html

Antes do SDLC, o processo de desenvolvimento de *software* era tomado como uma atividade informal sem regras, nem padrões. Com isto, a probabilidade de se originar vários problemas, tais como, atrasos no desenvolvimento, aumentos dos custos e baixa qualidade na solução criada, era consideravelmente mais elevada [125].

Na Figura 53, apresentada de seguida, estão apresentadas as sete etapas consideradas por Saini e Kaur, que serão descritas posteriormente [124].

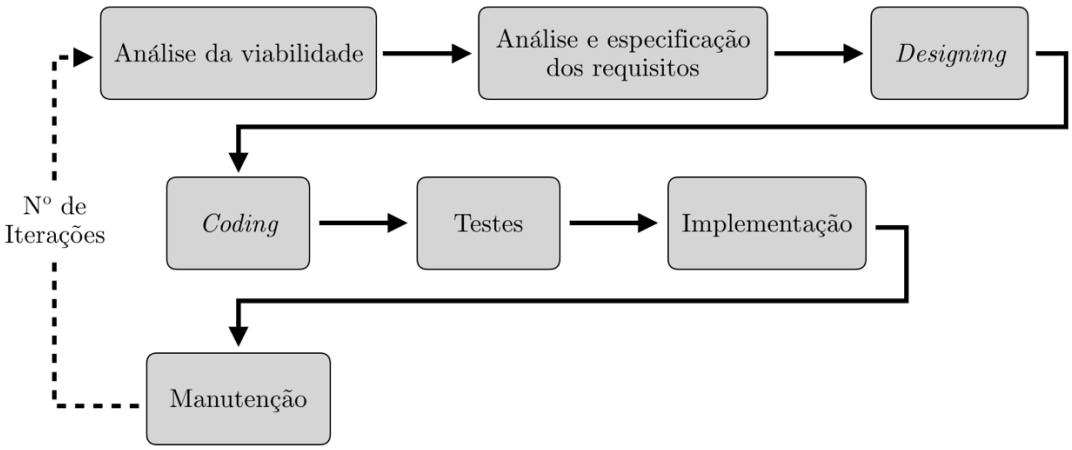


Figura 53 – Fases de Desenvolvimento de um software adaptado: A Review of Open Source Software Development Life Cycle Models (M. Saini e K. Kaur, 2014)

- **Análise de viabilidade:** nesta fase são analisados os dados de entrada e saída, processamento necessário, análise de custos e o planeamento do projeto. É ainda incluída a viabilidade técnica em termos de *software*, *hardware*, bem como pessoas qualificadas;
- **Análise e especificação de requisitos:** nesta fase são recolhidos e analisados os requisitos necessários para a elaboração da solução. No final desta análise, é espectável que todos os requisitos sejam conhecidos e explanados;
- **Desenho ou Conceção** (*Designing*)**:** Consiste na tradução dos requisitos especificados, na etapa anterior, numa estrutura lógica;
- **Programação** (*Coding*)**:** o desenvolvimento da solução é realizado nesta etapa. A lógica projetada, na etapa anterior, é traduzida em código de forma a ser posteriormente compilado;
- **Testes:** o código desenvolvido, na etapa anterior, é testado utilizando vários cenários de teste com o objetivo de corrigir e avaliar o sistema desenvolvido;

- **Implementação:** a solução desenvolvida é implementada para que possa ser disponibilizada ao utilizado para real uso. Pretende-se que o utilizado do sistema possa reportar erros ou problemas quando encontrados;
- **Manutenção:** o sistema desenvolvido poderá sofrer alterações de modo a corrigir alguns problemas reportados. Esta fase é responsável pela manutenção do sistema e também pelo correto funcionamento em versões posteriores.

Adicionalmente, e durante o desenvolvimento da componente prática desta dissertação foram criados dois repositórios Git[67], um para o desenvolvimento da REST API, e o segundo para a conceção da prova de conceito, a aplicação Android, apresentada no capítulo seguinte.

A particularidade de criar um repositório Git e de sincronizar o código desenvolvimento no próprio é a possibilidade de controlar diferentes versões do código desenvolvido, permitindo ainda realizar *backups* sistemáticos e um armazenamento seguro do mesmo. A utilização desta ferramenta mostrou-se particularmente útil em várias etapas no desenvolvimento da solução, uma vez que, permitiu recuperar antigas versões desenvolvidas.

[67] https://git-scm.com

CAPÍTULO **5**

PoC: Desenvolvimento de uma Aplicação Móvel em Android

O capítulo 5 está inteiramente reservado à apresentação da prova de conceito desenvolvida. A aplicação móvel idealizada para dispositivos Android, apresenta-se como um protótipo funcional da utilização da REST API desenvolvida.
Todos os detalhes relativos ao desenvolvimento serão explanados de seguida.

5.1. Motivação

A intenção de apresentar uma prova de conceito que usufruísse do desenvolvimento da REST API, surgiu aquando da elaboração da mesma.

Não estando definida nos objetivos iniciais da dissertação, a ideia tornou-se rapidamente indispensável, numa ótica pessoal, para fundamentar e apresentar as possibilidades futuras que o desenvolvimento de uma REST API permite.

5.2. Propósito da Aplicação

Tendo presente que o principal objetivo da dissertação é a recolha e o tratamento das informações relativas à oferta de viagens dos provedores presentes na solução, a aplicação idealizada fundamenta-se com o mesmo propósito – apresentar ao utilizador a viagem mais conveniente para o par de coordenadas providenciadas por si. De um modo semelhante, a prova de conceito idealizada, aproximar-se-á funcionalmente das aplicações móveis apresentadas em 2.5 – Análise de plataformas agregadoras de mobilidade, com uma UI/UX mais minimalista, face aos produtos já em mercado.

Para explicar o fluxo da aplicação, vai ser tida em consideração uma consulta realizada no dia 19 de Setembro de 2018, aproximadamente pelas 09h. Foram consideradas as cidades do Porto e de Coimbra, respetivamente, para partida e chegada da simulação da viagem.

A primeira interface apresentada ao utilizador – Figura 54 – é onde este terá a liberdade de introduzir, tanto as coordenadas de origem como de destino, da sua viagem.

De notar que, a pesquisa será realizada através da introdução de coordenadas, ao invés da introdução do nome da cidade, uma vez que um dos recursos exigidos para a construção da REST API, era a pesquisa através de coordenadas.

De notar também que, o mapa que constitui a interface, não apresentará uma rota entre as cidades, uma vez que como a solução aglomera vários provedores de serviços de mobilidade, a rota de cada um será distinta.

Assim, depois de introduzidas as coordenadas de origem e destino, os marcadores geográficos serão automaticamente adicionados e o zoom do mapa ajustado, para que seja possível visualizar ambos os marcadores no mapa.

Depois de introduzidas as coordenadas, é tanto possível realizar uma pesquisa genérica, isto é, sem a adição de filtros através do botão SEARCH, como aplicar métodos de ordenação, face aos resultados provenientes da consulta à REST API, pressionando o botão SORTING METHODS.

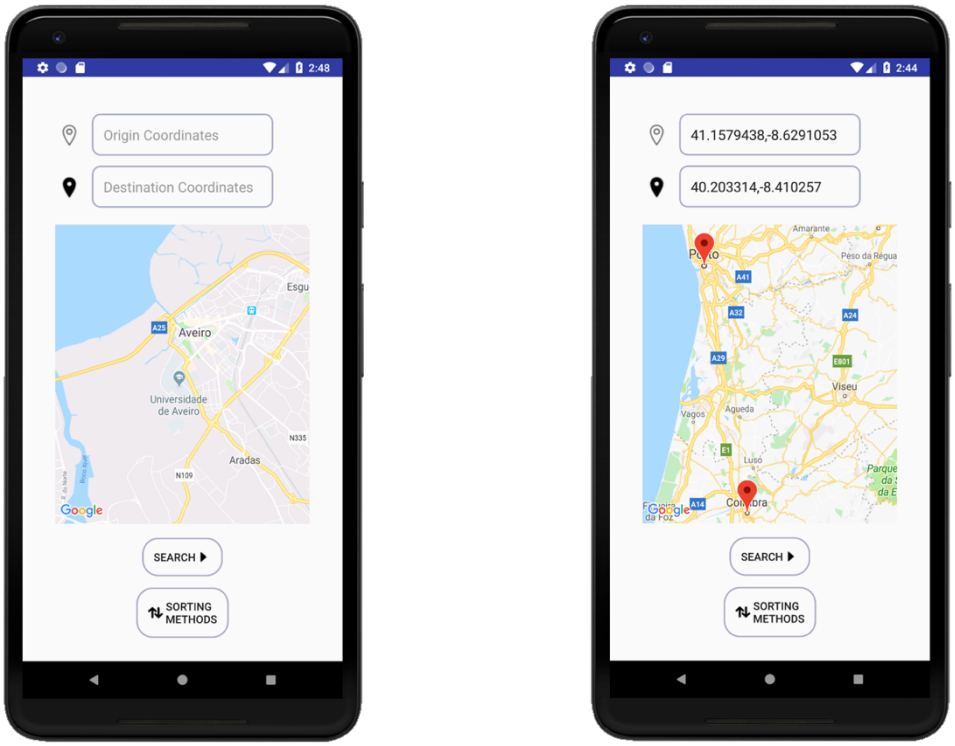


Figura 54 – Interface de Pesquisa por Coordenadas
À esquerda: Sem a introdução das coordenadas de origem e destino
À direita: Com a introdução das coordenadas de origem e destino

Para fins demonstrativos, será adotada a pesquisa com métodos de ordenação. Desta forma, ao pressionar o botão SORTING METHODS, será apresentado ao utilizador um primeiro *pop-up*, de forma a segmentar o tipo de ordenação que se pretende aplicar.

É tanto possível escolher que o resultado a apresentar seja ordenado através da ascensão do preço da viagem como pelo seu declínio, ou ainda a escolha prioritária do *provider* a apresentar no resultado.

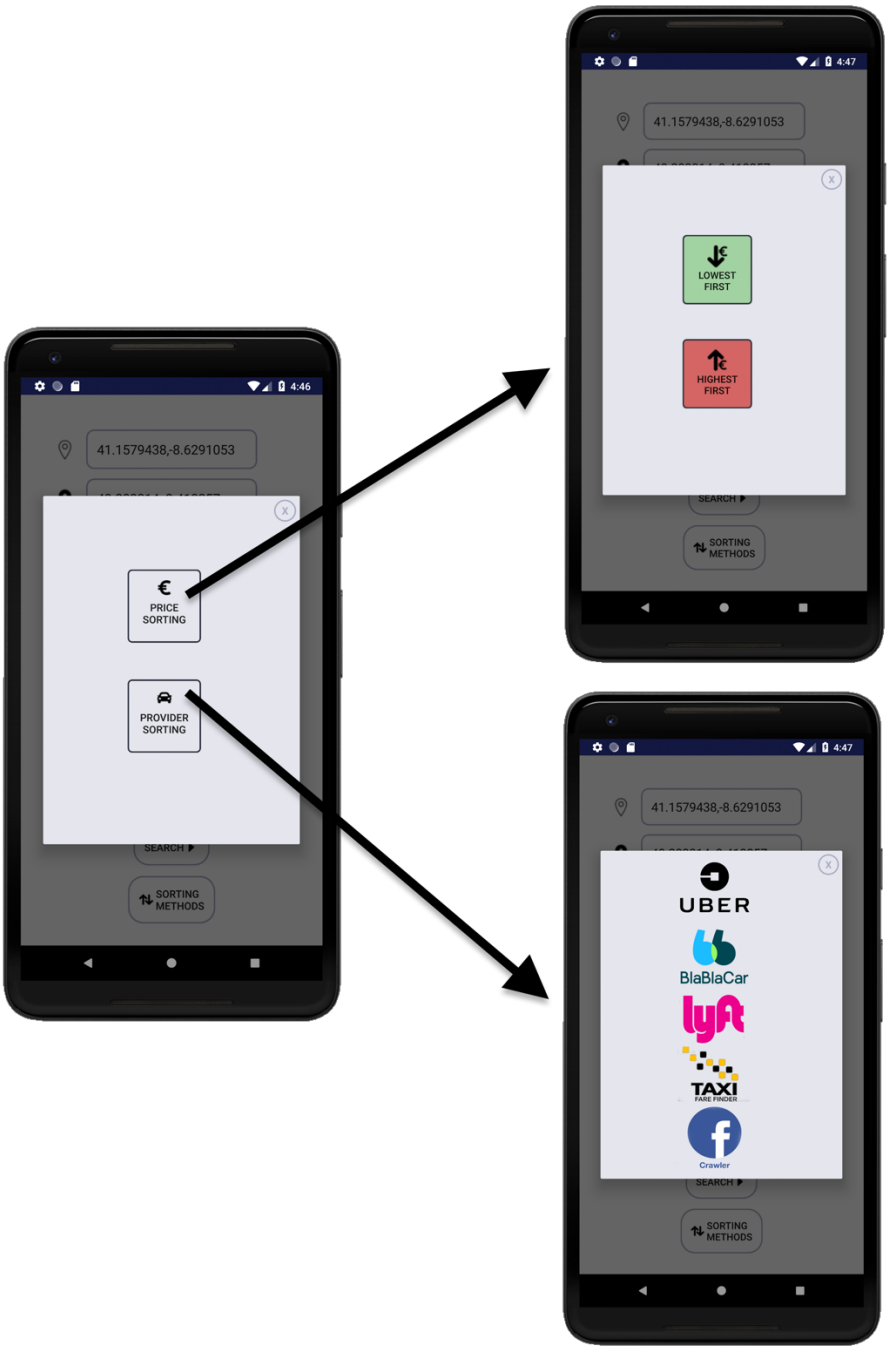


Figura 55 – Interfaces de Ordenação das Pesquisas

Na eventualidade de se pretender ordenar os resultados, cujo preço esteja organizado do mais acessível até ao mais caro, é selecionado o botão PRICE SORTING do primeiro *pop-up* e, posteriormente, escolhido o botão verde LOWEST FIRST. O resultado é ilustrado na Figura seguinte, Figura 56, onde é possível observar o preço associado a cada viagem nas informações das mesmas sobre a lateral direita do canto superior.

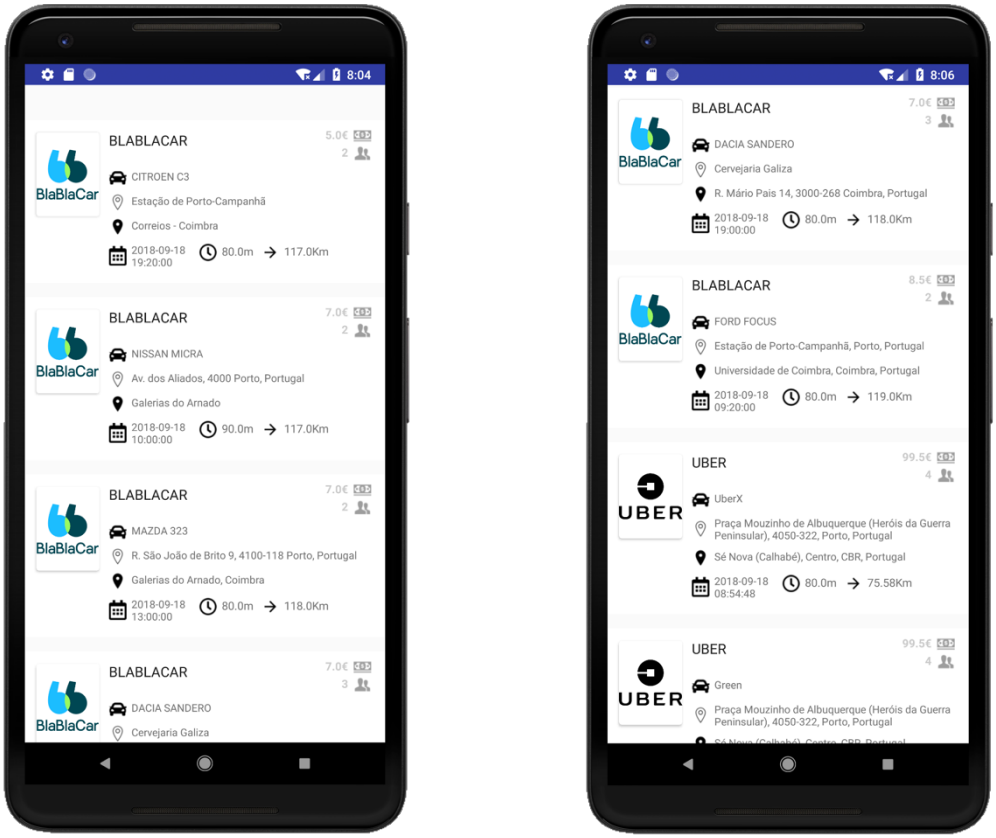


Figura 56 – Resultados da Pesquisa entre Porto e Coimbra, ordenado por preço ascendente

Para além do preço associado a cada viagem, outras informações compõem esta interface. No canto superior direito estão destacados tanto o preço como o número de lugares disponíveis para uma dada viagem. Estas informações foram estrategicamente colocadas, para uma rápida perceção e seleção da viagem pretendida, já que, depois da especificação da origem e destino, os critérios mais consideráveis para a seleção de uma viagem, serão o custo da mesma e o número de pessoas com as quais esta pode ser partilhada.

As restantes informações são apresentadas sobre o lado esquerdo, e indicam o nome do provedor do serviço, o veículo destinado para a deslocação, a morada detalhada ou um ponto de referência para os locais de partida e de chegada, bem

como a data e hora, duração em minutos da viagem e o número de quilómetros estimados.

Por último, e para acompanhar visualmente a informação, é adicionado, junto do nome do provedor do serviço, o seu logotipo.

Uma sumarização da organização dos elementos detalhados anteriormente, pode ser analisada na figura seguinte – Figura 57.

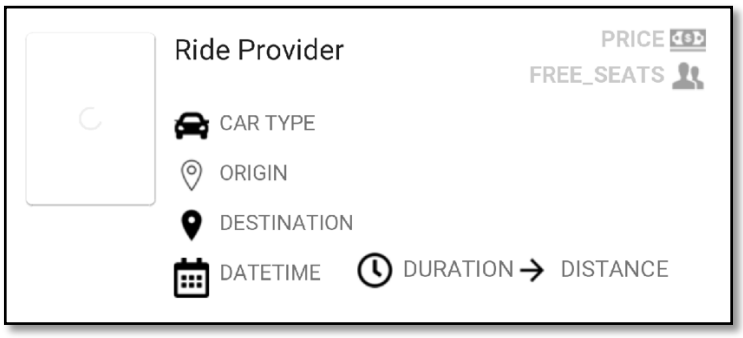


Figura 57 – Matriz genérica relativa à organização das informações de uma viagem

Contudo, podem existir ocasiões, nas quais sejam necessárias mais informações para além das apresentadas anteriormente. Um desses casos é quando se pretende apresentar as informações relativas a uma viagem que tenha sido extraída através do *crawler* que recolhe ofertas das redes sociais. Assim, para o exemplo de viagens provenientes da análise a grupos de Facebook, são adicionadas novas informações relativas ao anunciante da partilha, como o nome do mesmo, o corpo da mensagem que publicou – que pode conter informações que o interpretador não tenha detetado – e o grupo da qual a informação foi extraída.

A figura abaixo, sumariza assim, a organização dos novos elementos.

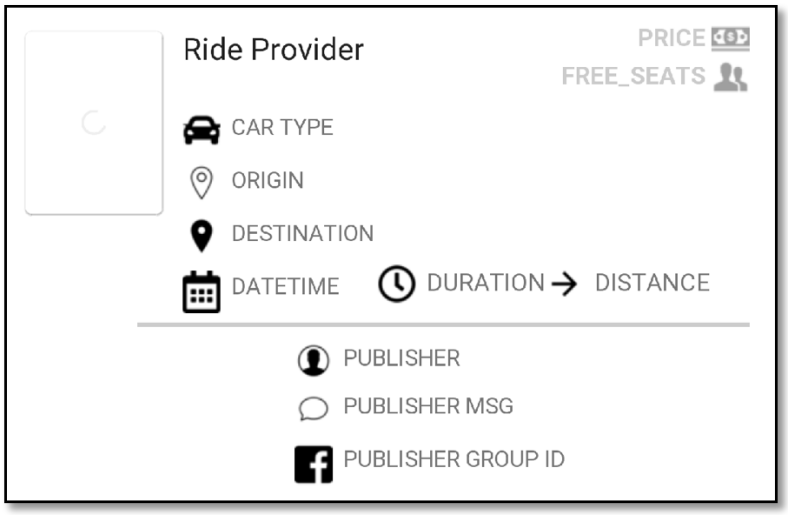


Figura 58 – Matriz relativa à organização das informações de uma viagem extraída do Facebook

De notar que, na eventualidade do provedor do serviço ser um serviço de *ridesharing*, onde os dados da viagem são introduzidos manualmente por um utilizador, por vezes, a informação de partida e de chegada, pode não ser a mais assertiva. O mesmo não acontece com os serviços de *carpooling*, onde existe uma conversão direta da coordenada de partida e de chegada na respetiva morada.

5.3. Desenvolvimento

Para o desenvolvimento da aplicação Android, foi utilizado o ambiente de desenvolvimento Android Studio[68] e criado um projeto com suporte para a versão mais recente do Android, a versão 8.1. Para o controlo de versões e gestão de dependências, foi utilizado o Gradle[69], e a Figura 59 apresenta as dependências utilizadas no projeto.

[68] https://developer.android.com/studio

[69] https://gradle.org

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'
    androidTestImplementation 'com.android.support.test.espresso:espresso-core:3.0.2'

    //RETROFIT
    implementation 'com.squareup.retrofit2:retrofit:2.4.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.4.0'
    implementation 'com.squareup.retrofit2:adapter-rxjava:2.1.0'
    implementation 'com.squareup.okhttp3:logging-interceptor:3.9.1'

    // DESIGN
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support:design:27.1.1'
    implementation 'com.android.support:recyclerview-v7:27.1.1'
    implementation 'com.android.support:cardview-v7:27.1.1'

    //LOADING AND CACHING IMAGES
    implementation 'com.github.bumptech.glide:glide:4.8.0'
    annotationProcessor 'com.github.bumptech.glide:compiler:4.8.0'

    // GOOGLE MAPS
    implementation 'com.google.android.gms:play-services-maps:15.0.1'
}
```

Figura 59 – Dependências utilizadas no projeto Android

5.3.1. Bibliotecas

Para que se consiga atingir o propósito apresentado anteriormente, foi necessária a utilização de bibliotecas que permitissem, sobretudo, uma correta conexão e extração dos conteúdos provenientes da REST API desenvolvida.

Retrofit2

O Retrofit[70], apresenta-se como uma biblioteca *HTTP Client*, disponível tanto para o desenvolvimento de aplicações móveis Android, como de soluções em Java, marcando pontos positivos pela simplicidade e desempenho que apresenta.

É através do Retrofit que é possível realizar conexões HTTP a *Web-Services* REST, onde são convertidos os vários *endpoints* disponíveis na solução, em interfaces Java. O resultado destas consultas, como já apresentado em capítulos anteriores, são as informações das viagens, expressas na notação JSON e é também, através da biblioteca, que será possível proceder à desserialização dos elementos pretendidos.

[70] https://square.github.io/retrofit

A definição do tipo de pedido pretendido (GET, POST, PUT, DELETE), bem como certos campos do *Header* – como é o caso do *access-token* – e os parâmetros de pesquisa especificados no URL da consulta, são definidos através de anotações, o que simplifica a compreensão e redução de código por parte do desenvolvedor. Todavia, o Retrofit não possuí um conversor JSON nativo para manipular os objetos JSON. Ao invés disso, utiliza convenientemente outras bibliotecas como é o caso do Gson[71].

```
@GET("rides/")
Call<MetaData> getRidesWithPagination(@Header("token") String token,
                                      @Query("origin") String origin,
                                      @Query("destination") String destination,
                                      @Query("page") int page);
```

Figura 60 – Exemplo do método GET ao endpoint rides

Paralelamente, o Retrofit utiliza ainda o OkHttp, também desenvolvido pela mesma companhia, de modo a operar com as solicitações de rede, sendo que esta é apresentada de seguida.

OkHttp

Como mencionado anteriormente, o OkHttp[72] é uma biblioteca que é utilizada pelo Retrofit e que permite a realização de pedidos HTTP. Estes pedidos podem ser personalizados, sendo assim possível adicionar elementos do *Header* ou um *interceptor*, de modo a acompanhar os *logs* relativos a todos os pedidos e respostas ao *endpoint* pretendido – destacado a amarelo na Figura 61.

Através da criação de um `OkHttpClient` e da associação deste ao cliente Retrofit, onde é especificado o URL do *endpoint* desejado e o conversor desejado de modo a transformar as respostas – neste caso em JSON – em objetos, é possível processar as respostas provenientes da API, identificado a laranja na Figura 61.

[71] https://github.com/google/gson

[72] http://square.github.io/okhttp

```
private void init (@NonNull String url){
    OkHttpClient.Builder httpClient = new OkHttpClient.Builder();
    HttpLoggingInterceptor interceptor = new HttpLoggingInterceptor();
    interceptor.setLevel(HttpLoggingInterceptor.Level.BODY);
    httpClient.connectTimeout( timeout: 30, TimeUnit.SECONDS);
    httpClient.readTimeout( timeout: 30, TimeUnit.SECONDS);
    httpClient.addInterceptor(interceptor);
    httpClient.addInterceptor(new Interceptor() {
        @Override
        public Response intercept(@NonNull Interceptor.Chain chain) throws IOException {
            Request original = chain.request();
            Request request = original.newBuilder()
                    .header( name: "Content-Type",  value: "application/json")
                    .method(original.method(), original.body())
                    .build();
            return chain.proceed(request);
        }
    });

    Retrofit.Builder builderUUID = new Retrofit.Builder()
            .baseUrl(url)
            .client(httpClient.build())
            .addCallAdapterFactory(RxJavaCallAdapterFactory.create())
            .addConverterFactory(GsonConverterFactory.create(new GsonBuilder().disableHtmlEscaping().create()));
    Retrofit retrofit = builderUUID.build();
```

Figura 61 – Exemplo de Criação de um cliente Retrofit e OkHTTPClient

CAPÍTULO **6**

Testes à API

Para que fosse possível avaliar o tempo que a solução desenvolvida demora a responder aos pedidos que lhe são solicitados, foram elaborados testes de carga.

A ferramenta utilizada que deu de suporte à criação dos testes realizados foi o Apache JMeter[73]. Esta é uma aplicação desktop, *open-source*, desenvolvida em Java, que ofere suporte a qualquer sistema operativo, onde se espera que o comportamento funcional seja o mesmo, independentemente do sistema, desde que o Java Development Kit – JDK – esteja instalado nas mesmas [126].

Desenvolvido para executar testes funcionais e para medir o desempenho de aplicações, o JMeter foi originalmente projetado para testar aplicações *Web*, expandindo-se, ao longo dos anos, para outros tipos de testes, como de *performance*, carga e *stress*. O objetivo estratégico do JMeter é prover cenários de testes realistas, simulando, o mais possível a realidade de utilização da solução desenvolvida.

Uma vez que a solução agregadora concebida, realiza vários pedidos às APIs dos *providers* registados, recolhe e processa as suas respostas, regista-as na base de dados e devolve uma nova resposta ao utilizador com base nas informações recolhidas, o estudo relativo ao tempo deste processo é particularmente interessante de ser apresentado.

[73] https://jmeter.apache.org

6.1. Criação dos Testes

O primeiro conceito na criação dos testes são os utilizadores, onde estes são representados por *threads*, e estas constituídas por etapas que são executadas sequencialmente, num *script* onde são planificados os testes.

Cada etapa executada por uma *thread*, como por exemplo, a interação com páginas *Web*, é realizada através de um *listener*. Resumidamente, o *script* é constituído por uma sequência de *listeners* agrupados, de forma a seguir uma ordem lógia de utilização.

O passo seguinte é a recolha das informações que foram recolhidas da execução dos *listeners*, para que seja assim possível a análise de métricas, como o tempo médio, mínimo e máximo de resposta e o *throughtput*, conhecido como o número de pedidos por minuto que o servidor consegue processar.

Por último, na composição de um *script*, os *listeners* são agrupados dentro de *Thread Groups*, que irão determinar o comportamento das *threads*, entenda-se, utilizadores. Nestes, são configurados a quantidade de utilizadores que estão a ser simulados e a duração da execução do *script*.

A figura abaixo, representa a estrutura e organização dos elementos apresentados anteriormente.

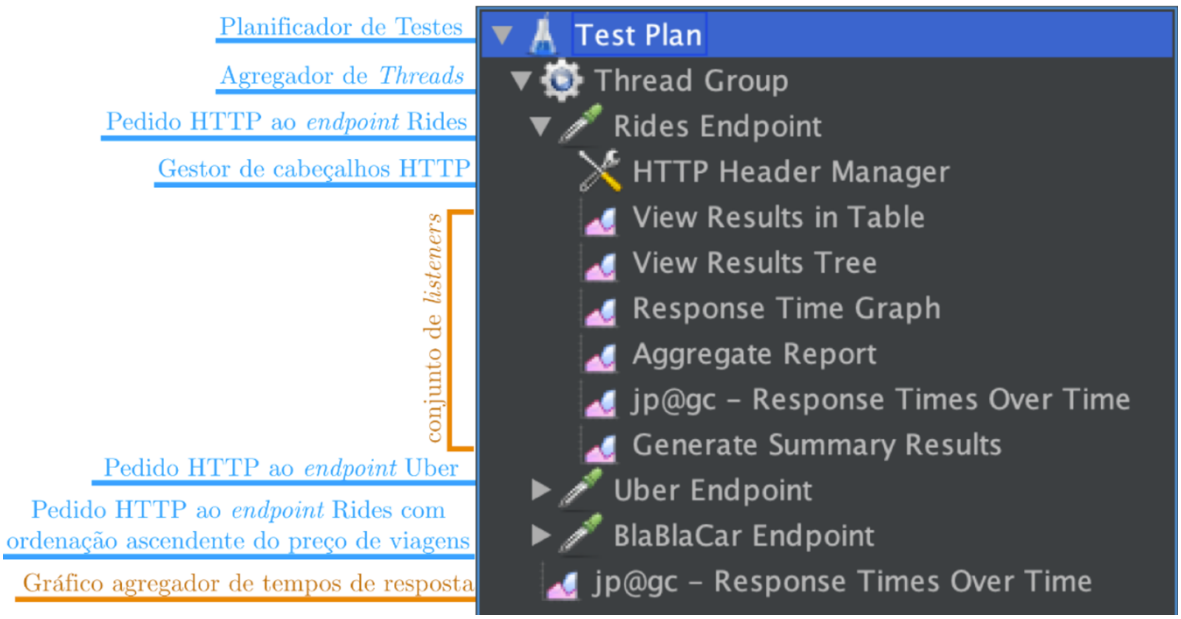


Figura 62 – Elementos do Test Plan

Assim, e para que o teste seja realizado, é necessário proceder à configuração do mesmo.

Primeiramente, no *Thread Group*, são expressos o número de utilizadores que consultam um determinado *endpoint* e o número de pedidos por si solicitados.

O *Thread Group* é assim constituído por três pedidos HTTP, como apresentado na Figura 62 e para cada um destes, é necessário especificar o tipo de protocolo, o IP ou nome do servidor, a porta – se aplicável –, o método HTTP, o *endpoint* desejável e eventuais parâmetros presentes no URL do pedido, como exemplificado na Figura seguinte.

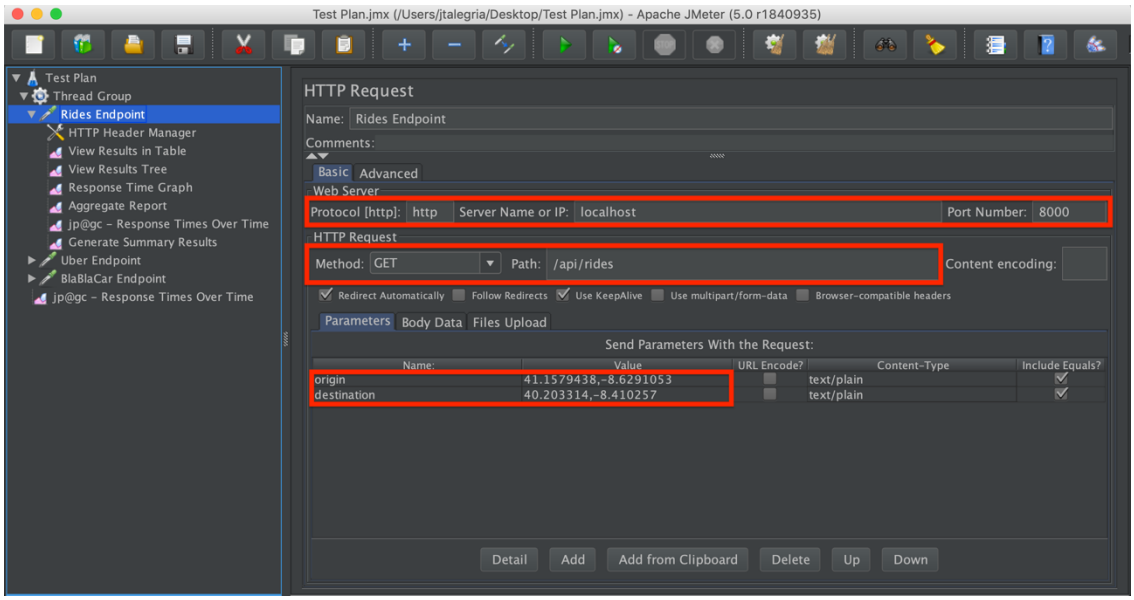


Figura 63 – Especificação dos parâmetros de um pedido HTTP

Para a introdução do *access-token* no cabeçalho do pedido, é adicionado um HTTP Header Manager, que torna possível a introdução dos parâmetros pretendidos.

Os *listeners* associados ao pedido HTTP, irão tanto apresentar as respostas associados a cada pedido efetuado, bem como um resumo das métricas associadas ao conjunto de respostas recebidas.

6.2. Resultado dos Testes

Foram realizados dois testes distintos, todos com o propósito de avaliar o tempo de resposta e o número de respostas, por minuto, que o servidor consegue aguentar. De notar que, para a realização dos testes, a conversão das coordenadas nas respetivas moradas, foi desativada, uma vez que esse serviço é limitado a um dado número de pedidos por dia, na versão *Free*, o que resultaria assim em possíveis exceções durante a realização dos pedidos e, por consequência, em respostas com o código 400 – Bad Request.

O primeiro teste consiste em simular 100 pedidos ao *endpoint* Rides, onde são verificadas as viagens disponíveis nos diferentes providers, para os pares de coordenadas inseridos. Este teste foi realizado 5 vezes, de modo a realizar uma média, relativamente às métricas de cada conjunto de respostas. Os gráficos resultantes, relativos às respostas recebidas, em função do tempo, são apresentados de seguida.

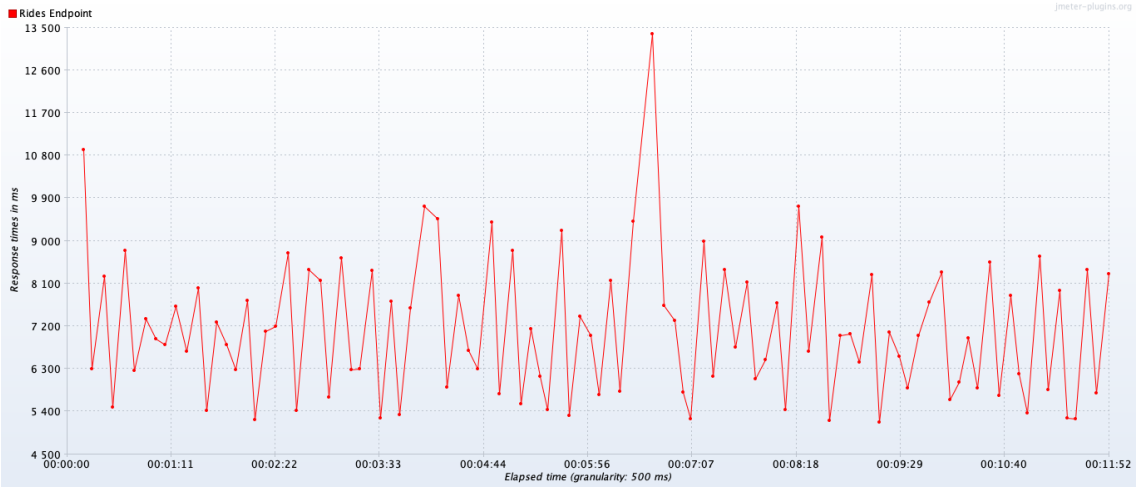


Figura 64 – Teste#1 - Tempos de resposta ao Endpoint Rides

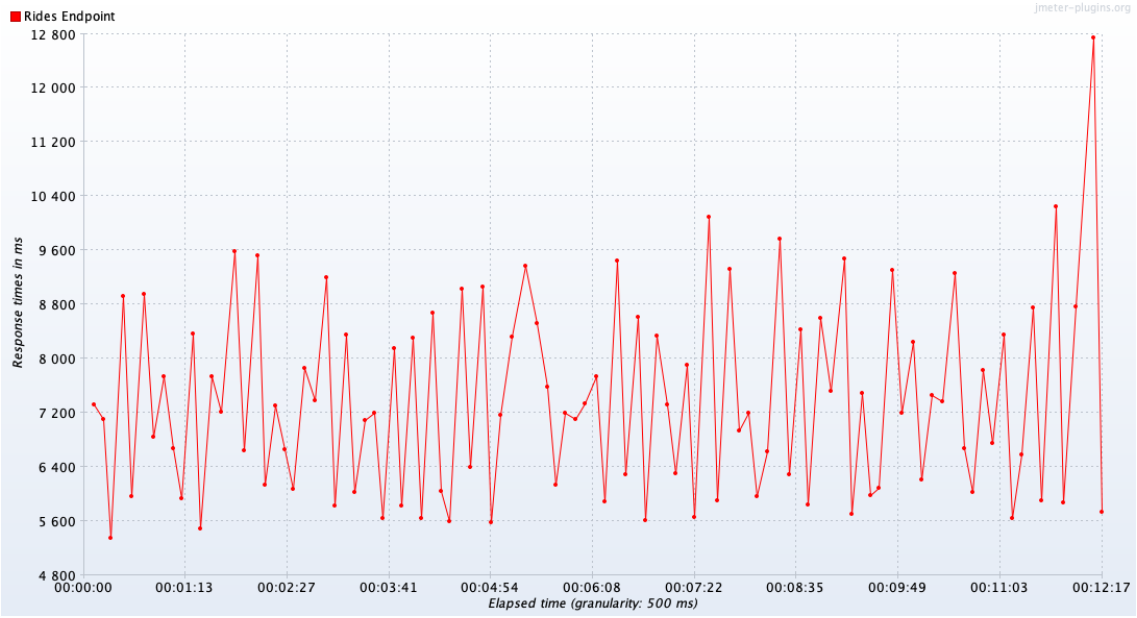


Figura 65 – Teste#2 - Tempos de resposta ao Endpoint Rides

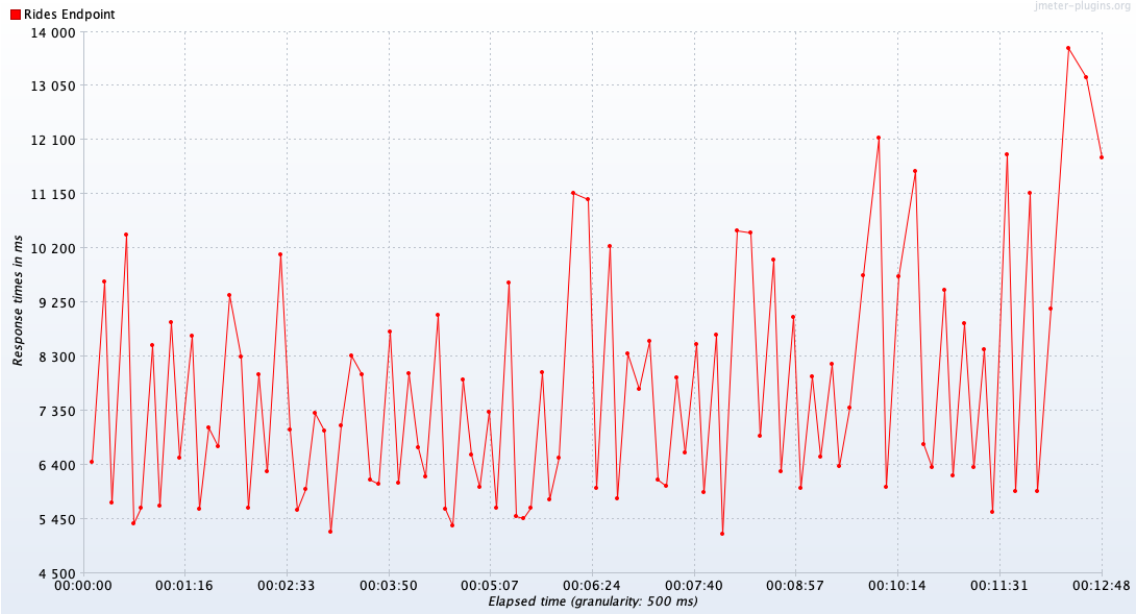


Figura 66 – Teste#3 - Tempos de resposta ao Endpoint Rides

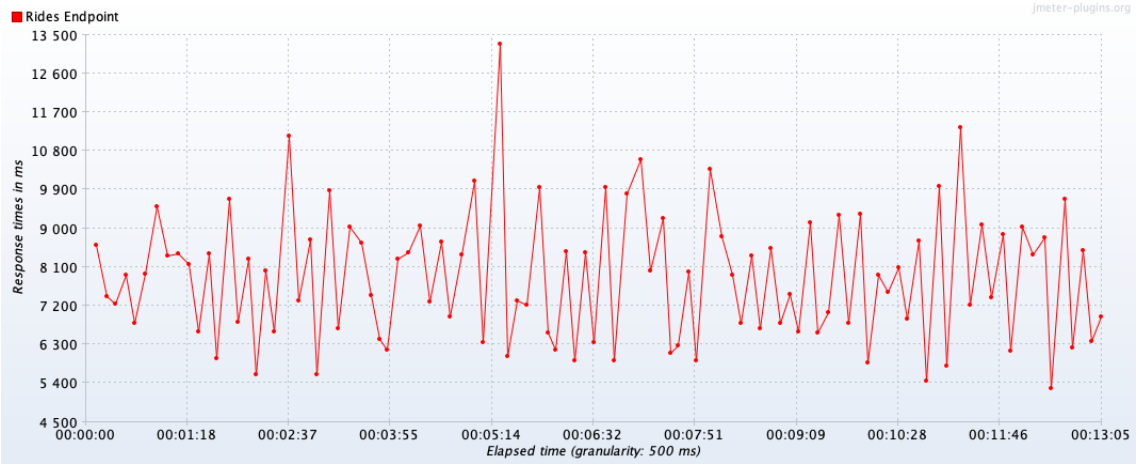


Figura 67 – Teste#4 - Tempos de resposta ao Endpoint Rides

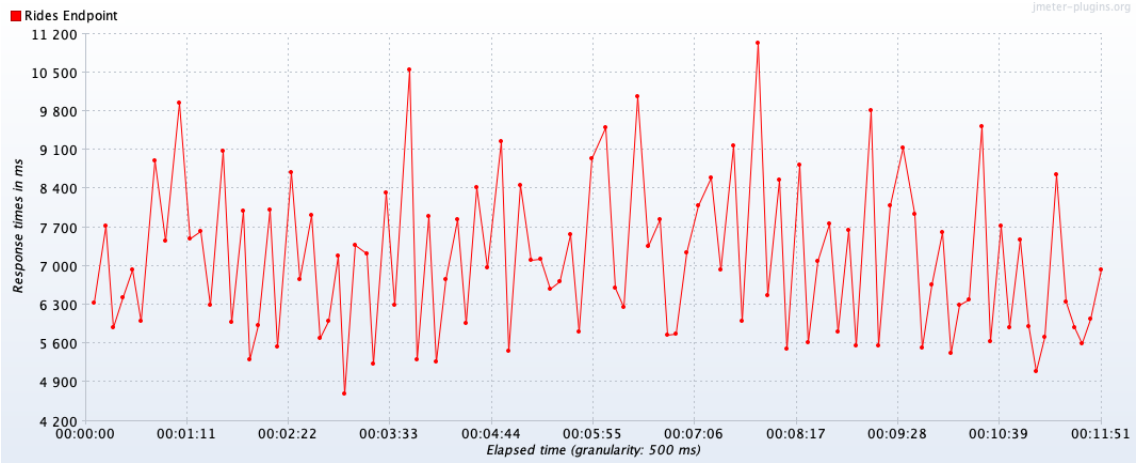


Figura 68 – Teste#5 - Tempos de resposta ao Endpoint Rides

A tabela seguinte, resume os resultados obtidos dos 5 testes elaborados, e a média dos mesmos.

Tabela 29 – Comparação das métricas relativas aos testes ao Endpoint Rides

	Número de Amostras	**Média (ms)**	**Mínimo (ms)**	**Máximo (ms)**	***Throughput* (respostas/minuto)**	**Tempo Decorrido (m)**
Teste 1	100	7112	5177	13364	8,4	11,52
Teste 2	100	7364	5352	12744	8,1	12,17
Teste 3	100	7675	5188	13721	7,8	12,48
Teste 4	100	7848	5276	13277	7,6	13,05
Teste 5	100	7105	4688	11041	8,4	11,51
MÉDIA	100	7420,8	5136,2	12829,4	8,06	12,146

Deste estudo, é possível observar que a média de respostas por minuto, ao *endpoint* Rides, é cerca de 8 segundos, e o tempo médio de resposta a cada pedido é cerca de 7,4 segundos, num tempo total médio de 12 minutos e 15 segundos, respetivamente.

Pontualmente, existem picos onde o tempo de resposta é consideravelmente mais alto, possivelmente porque um ou mais *providers* podem apresentar um maior volume de pedidos num dado intervalo de tempo, ou apresentar mecanismos de segurança que limitem, ou atrasem a resposta de um pedido, quando enfrentam um elevado número de pedidos.

O teste seguinte compara dois *providers* em específico. Neste, são considerados dois pedidos HTTP distintos, um ao *endpoint* Uber e outro ao *endpoint* BlaBlacar, ambos executados 100 vezes. O gráfico da Figura 69, apresenta os tempos de resposta de ambos.

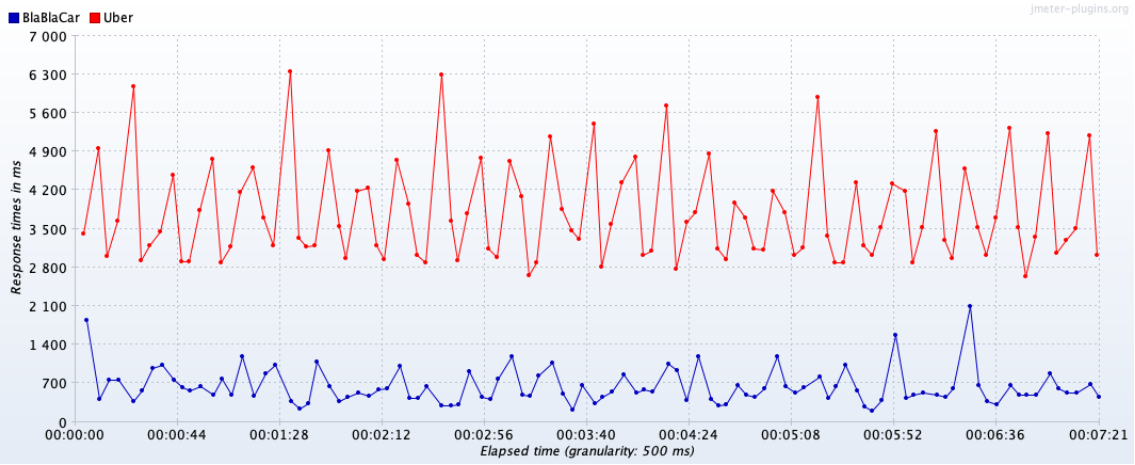


Figura 69 – Tempos de resposta aos Endpoints Uber e BlaBlaCar

A tabela seguinte, resume os resultados obtidos do teste elaborado.

Tabela 30 – Métricas relativas ao teste aos Endpoints Uber e BlaBlaCar

	Número de Amostras	**Média (ms)**	**Mínimo (ms)**	**Máximo (ms)**	***Throughput* (respostas/minuto)**
Uber	100	3764	2646	6350	13,6
BlaBlaCar	100	639	212	2099	13,7

Analisando os resultados obtidos, é rapidamente observável que os tempos de resposta, por parte do *provider* BlaBlaCar, são consideravelmente mais baixos em comparação aos do *provider* Uber.

Através deste resultado, é constatável que a consulta a diferentes *providers* não apresenta o mesmo tempo de resposta, pelo que a consulta ao *endpoint* Rides, apresentará sempre tempos de resposta mais elevados face a outro *endpoint* que apenas consulte um dado *provider*.

CAPÍTULO 7

Conclusões

Com o desenvolvimento da solução apresentada nos capítulos anteriores, foi possível criar um agregador dos principais provedores de serviços atualmente existentes.

Após o estudo dos serviços apresentados por cada um desses provedores e de aplicações similares já existentes, foram avaliados os prós e contras das mesmas, de modo a enriquecer a solução desenvolvida.

O trabalho consistiu no desenvolvimento de uma REST API, através do Django Rest Framework, responsável por se conectar aos diferentes *providers*, recolher as informações das viagens oferecidas, para um dado par de coordenadas, como o preço, data e hora de partida, duração e distância de uma dada viagem, e com estas, gerar um modelo genérico de resposta, na notação JSON.

Por fim, a ferramenta desenvolvida foi assim validada numa prova de conceito, onde foi realizada uma aplicação Android que se conecta, recolhe e apresenta os dados resultantes das consultas à REST API.

7.1. Trabalho Desenvolvido no Estágio

Os principais objetivos deste projeto, relacionavam-se com a análise das plataformas de mobilidade alternativas já existentes, com o desenvolvimento de um agregador que reunisse o maior número das plataformas previamente estudadas, com a integração das viagens partilhadas nas redes sociais, onde de uma investigação realizada, o Facebook apresentou uma maior adesão por parte dos utilizadores, razão pela qual foi escolhida para integrar a solução da API REST.

Com o desenvolvimento deste projeto, foi possível constatar que, o número de soluções alternativas ao uso do veículo próprio, como é o caso do *carsharing* e *ridesharing*, têm tido, nos últimos anos, um elevado crescimento, contribuindo tanto para o crescimento de soluções para as cidades inteligentes, proporcionando uma grande oferta para os seus adeptos, como para a diminuição das emissões de poluentes atmosféricos.

Contudo, e com o desenvolvimento desta plataforma agregadora, a procura de viagens pelos *providers* existentes, torna-se num processo centralizado, proporcionando ao utilizador uma redução no tempo de procura do melhor serviço para as suas necessidades.

Ao realizar a presente dissertação num contexto empresarial, o auxílio e o esclarecimento de dúvidas no desenvolvimento deste projeto, era praticamente imediato, o que proporcionava mais agilidade no desenvolvimento do projeto. Para além de ter sido útil no esclarecimento de dúvidas, o contexto empresarial referido anteriormente, permitiu ainda o contacto com novas tecnologias, a oportunidade de integração nos processos de trabalho da empresa, bem como num grande enriquecimento a nível pessoal.

7.2. Integração nos Processos de Trabalho da empresa

Para a execução deste projeto, foram utilizadas as mesmas ferramentas que são empregues em projetos reais da empresa, o que permitiu obter uma melhor perceção do trabalho num âmbito empresarial, através da integração nos processos de trabalho empregues na Ubiwhere.

Relativamente ao armazenamento do código correspondente ao desenvolvimento da API REST agregadora dos serviços de mobilidade alternativa, bem como da aplicação Android, apresentada anteriormente como prova de conceito da solução desenvolvida, foi utilizado o GitLab[74] como repositório, e para o controlo de versões, foi utilizado o SourceTree[75], apresentado na Figura 70.

Figura 70 – Controlo de versões com o SourceTree

[74] https://about.gitlab.com
[75] https://www.sourcetreeapp.com

No que diz respeito à gestão do projeto, planeamento e organização das tarefas relativas ao mesmo, foi utilizado o Redmine[76], e os quadros presentes nos repositórios do Gitlab, onde estão contidas as *issues* associadas ao projeto, podem ser observadas na Figura 71.

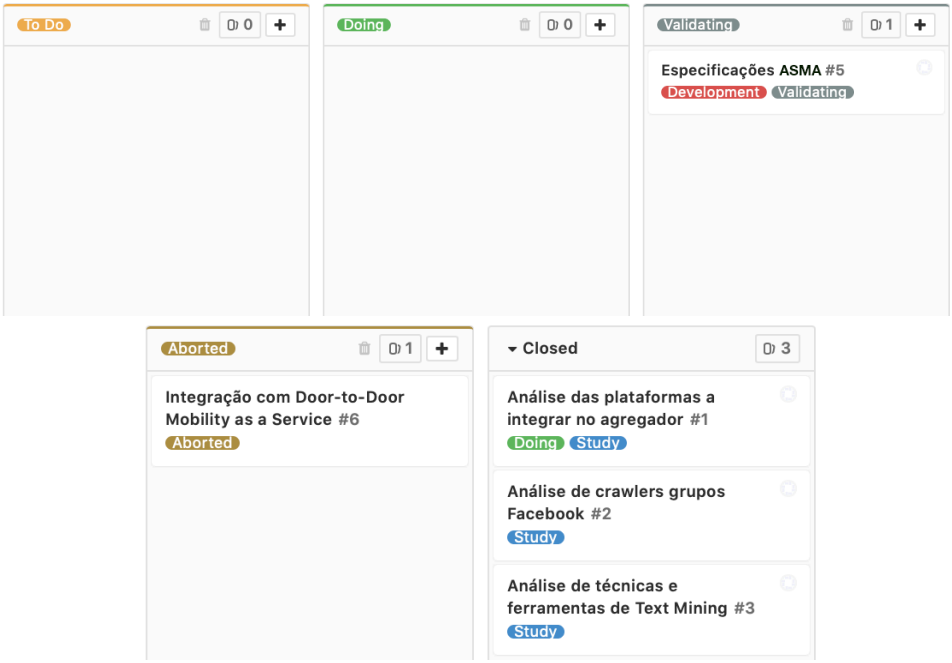


Figura 71 – Board de tarefas relativas ao desenvolvimento da API REST

7.3. Lições Aprendidas

Com a realização deste estágio, foi possível adquirir uma melhor perceção sobre como será trabalhar num contexto empresarial, que difere do contexto académico em diversos aspetos.

Enquanto que no contexto universitário, é importante demonstrar responsabilidade para obter bons resultados, no contexto empresarial, o tipo de responsabilidade que deve ser demonstrada é um pouco diferente, uma vez que a falta de responsabilidade não só terá impacto nos resultados individuais, mas também nos resultados de toda a equipa, podendo ainda afetar o cliente e a imagem com que este ficará de toda a equipa da empresa.

[76] https://www.redmine.org

Assim, tal como no meio académico é essencial saber trabalhar em equipa, aceitar críticas e aprender com as mesmas é igualmente importante. Autonomia e espírito crítico são também qualidades indispensáveis, para que seja possível proceder à divisão de tarefas pelos elementos da equipa de forma equilibrada.

As capacidades de comunicação são também importantes, não só no que se refere ao trabalho em equipa, contribuindo para a criação de um bom ambiente entre colegas e para que seja possível alcançar melhores resultados, mas também no que diz respeito à comunicação com clientes. Uma boa comunicação permite que sejam evitados erros no levantamento de requisitos dos projetos, por exemplo, ou outros tipos de erros resultantes de uma má articulação dos pensamentos ou de uma incorreta interpretação dos mesmos.

Para além disso, é vantajoso saber organizar o tempo e as tarefas, sobretudo quando se trabalha em várias *issues*, tendo em conta os prazos estipulados para a entrega dos produtos aos clientes.

7.4. Evolução e Trabalho Futuro

Para além das funcionalidades desenvolvidas, a integração de novos provedores à solução, enriqueceria a mesma e aumentaria a oferta de alternativas de mobilidade possíveis de serem integradas em projetos futuros. Parte dos provedores estudados anteriormente não foram integrados, uma vez que só aceitavam a integração dos seus serviços a terceiros com um custo associado, ou recusavam o acesso por este projeto ter sido desenvolvido para fins académicos.

A manutenção da ferramenta é outro cuidado a ter, já que podem ocorrer alterações nos serviços prestados pelos *providers*, que constituem esta solução agregadora, pelo que o suporte à solução desenvolvida é uma responsabilidade a manter.

7.5. Dificuldades Sentidas

No decorrer do projeto, surgiram algumas dificuldades que tiveram de ser resolvidas e ultrapassadas, para a correta concretização do mesmo.

A primeira dificuldade enfrentada foi o desenvolvimento inicial da API REST, uma vez que, até então, não tinha surgido a oportunidade de desenvolver uma.

Por outro lado, a utilização do Django Rest Framework, e por conseguinte, a linguagem Python, permitiram o desenvolvimento da mesma mais facilmente, uma vez que, de um ponto vista pessoal, já tinha desenvolvido outros projetos utilizando a *framework* Django e já apresentava conhecimentos relativos à intrínseca arquitetura MVT. Deste modo, e com as instruções retidas do estado de arte elaborado, as restantes dificuldades foram superadas, consultando a documentação do DRF existente [127].

Outra dificuldade sentida, principalmente por não ter até então contacto, foi o desenvolvimento e planeamento de tarefas automatizadas. Consultando algumas críticas *online*, rapidamente foram encontradas várias soluções para o desenvolvimento de tarefas assíncronas e para o agendamento de tarefas, diretamente no projeto.

Por outro lado, e aquando do desenvolvimento do interpretador das viagens publicas na rede social Facebook, a principal dificuldade sentida foi na conceção das expressões regulares, que conseguissem abranger as inúmeras formas de como os utilizadores da rede, normalmente, interagem e realizam as suas publicações.

Por último, convém também mencionar a dificuldade vivenciada aquando do contacto com alguns provedores de serviço de mobilidade. Muitos dos provedores apresentados na análise realizada no Estado de Arte, limitaram o acesso aos serviços, condicionando assim parte do desenvolvimento da solução, quer por negarem o acesso a estudantes, quer por apresentarem custos impossíveis de suportar para o desenvolvimento desta prova de conceito.

Referências

[1] L. Willi, "The State of European Car-Sharing - Final Report D 2.4 Work Package 2," 2010.

[2] Texas A&M Transportation Institue, "Traffic Gridlock Sets New Records for Traveler Misery," 2015. [Online]. Available: https://mobility.tamu.edu/ums/media-information/press-release/.

[3] J. Alonso-Mora *et al.*, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proc. Natl. Acad. Sci.*, 2017.

[4] KBB, "Car-sharing: conheça as plataformas disponíveis em Portugal," 2017. [Online]. Available: https://www.kbb.pt/detalhes-noticia/car-sharing-plataformas-disponiveis-portugal/?ID=599. [Accessed: 25-Sep-2018].

[5] R. P. Payyanadan and J. D. Lee, "Understanding the ridesharing needs of older adults," *Travel Behav. Soc.*, vol. 13, no. June, pp. 155–164, 2018.

[6] Y. Wang, S. Winter, and M. Tomko, "Collaborative activity-based ridesharing," *J. Transp. Geogr.*, vol. 72, pp. 131–138, 2018.

[7] M. Furuhata, M. Dessouky, F. Ordóñez, M. E. Brunet, X. Wang, and S. Koenig, "Ridesharing: The state-of-the-art and future directions," *Transp. Res. Part B Methodol.*, vol. 57, pp. 28–46, 2013.

[8] R. Vosooghi *et al.*, "A Critical Analysis of Travel Demand Estimation for New One-Way Carsharing Systems," pp. 199–205, 2017.

[9] S. Shaheen, D. Sperling, and C. Wagner, "A Short History of Carsharing in the 90's," 1999.

[10] M. Namazu and H. Dowlatabadi, "Vehicle ownership reduction: A comparison of one-way and two-way carsharing systems," *Transp. Policy*, vol. 64, no. November 2016, pp. 38–50, 2018.

[11] NOCTULA, "ECO.mob – Programa de Mobilidade Sustentável." [Online]. Available: http://noctula.pt/eco-mob-programa-de-mobilidade/. [Accessed: 21-Dec-2017].

[12] J. C. Derisio, *Introdução ao controle de poluição ambiental*. Oficina de textos, 2012.

[13] T. Litman, "Evaluating Carsharing Benefits," 2015.

[14] APORFEST, "Ridesharing: à boleia para os festivais!," 2015. [Online]. Available: http://www.aporfest.pt/single-post/2015/11/16/Ridesharing-à-boleia-para-os-festivais. [Accessed: 21-Dec-2017].

[15] Público, "Portugal eleito pela primeira vez melhor destino europeu nos 'óscares' do Turismo," 2017. [Online]. Available: https://www.publico.pt/2017/09/30/fugas/noticia/portugal-eleito-pela-primeira-vez-melhor-destino-europeu-nos-oscares-do-turismo-1787290. [Accessed: 18-Dec-2017].

[16] Uber Technologies Inc., "Uber Developers: Introduction to Riders." [Online]. Available: https://developer.uber.com/docs/riders/introduction. [Accessed: 17-Nov-2017].

[17] Uber Technologies Inc., "Uber Newsroom: Fatos e Dados sobre a Uber," 2017. [Online]. Available: https://www.uber.com/pt-BR/newsroom/fatos-e-dados-sobre-uber/. [Accessed: 07-Dec-2017].

[18] E-konomista, "Uber vs. Cabify: qual a melhor opção?" [Online]. Available: http://www.e-konomista.pt/artigo/uber-vs-cabify-qual-a-melhor-opcao/. [Accessed: 26-Nov-2017].

[19] J. P. Pullen, "Everything You Need to Know About Uber," 2014. [Online]. Available: http://time.com/3556741/uber/. [Accessed: 17-Nov-2017].

[20] Softonic International S.A., "Waze, o que é isso? O aplicativo de GPS que faz milagres…." [Online]. Available: https://www.softonic.com.br/artigos/waze-como-funciona. [Accessed: 26-Nov-2017].

[21] M. Bruzadin, "Como é andar com o Cabify, o novo concorrente do Uber na cidade de São Paulo," 2016. [Online]. Available: http://www.melhoresdestinos.com.br/cabify-teste-concorrente-uber.html. [Accessed: 26-Nov-2017].

[22] E. Betters, "What is Uber and how does it work?," 2016. [Online]. Available: http://www.pocket-lint.com/news/139559-what-is-uber-and-how-does-it-work. [Accessed: 17-Nov-2017].

[23] Uber Technologies Inc., "Opções de Conta e Pagamento: Atualizar uma forma de pagamento na sua conta." [Online]. Available: https://help.uber.com/pt_PT/h/8f78dca4-9d75-44f1-bdc1-e90ca3da0319. [Accessed: 17-Nov-2017].

[24] Uber Technologies Inc., "Uber: Opções de Conta e Pagamento - Pagar com dinheiro." [Online]. Available: https://help.uber.com/pt_PT/h/ba02bcb0-4bdc-417a-a236-8fe1582adffc. [Accessed: 17-Nov-2017].

[25] Uber Technologies Inc., "Conheça os serviços Uber disponíveis em Portugal." [Online]. Available: https://get.uber.com/p/uber-portugal-parceiros-servicos/. [Accessed: 17-Nov-2017].

[26] P. Gil, "How Uber Works and the Pros and Cons," 2017. [Online]. Available: https://www.lifewire.com/how-does-uber-work-3862752. [Accessed: 17-Nov-2017].

[27] Uber Technologies Inc., "O que oferecemos," 2018. [Online]. Available: https://www.uber.com/pt-PT/about/uber-offerings/. [Accessed: 26-Sep-2018].

[28] Uber Technologies Inc., "Como são calculados os preços?," 2018. [Online]. Available: https://help.uber.com/h/d2d43bbc-f4bb-4882-b8bb-4bd8acf03a9d. [Accessed: 26-Sep-2018].

[29] Vida Cigana, "O que é e como funciona a Cabify, app rival da Uber," 2017. [Online]. Available: https://vidacigana.com/como-funciona-cabify/. [Accessed: 26-Nov-2017].

[30] Circula Seguro, "UberEats – uma plataforma inovadora? Como funciona?," 2018. [Online]. Available: http://www.circulaseguro.pt/mobilidade-sustentavel/ubereats-plataforma-inovadora-funciona. [Accessed: 26-Sep-2018].

[31] Tecmundo, "Uber Freight: empresa lança app para caminhoneiros nos EUA," 2017. [Online]. Available: https://www.tecmundo.com.br/uber/116872-uber-freight-empresa-lanca-app-caminhoneiros-eua.htm. [Accessed: 26-Sep-2018].

[32] ProgrammableWeb, "Uber API." [Online]. Available: https://www.programmableweb.com/api/uber. [Accessed: 17-Nov-2017].

[33] Uber Technologies Inc., "Uber Developers: Python SDK." [Online]. Available: https://developer.uber.com/docs/riders/ride-requests/tutorials/api/python. [Accessed: 23-Nov-2017].

[34] Uber Technologies Inc., "Uber Developers: Introduction to the API." [Online]. Available: https://developer.uber.com/docs/riders/ride-requests/tutorials/api/introduction. [Accessed: 23-Nov-2017].

[35] Razão Automóvel, "Cabify: concorrente da Uber já chegou a Portugal," 2015. [Online]. Available: https://www.razaoautomovel.com/2016/05/cabify-concorrente-da-uber-ja-chegou-a-portugal. [Accessed: 26-Nov-2017].

[36] Cabify, "Cabify: Em que cidades opera a Cabify?" [Online]. Available: https://help.cabify.com/hc/pt/articles/115000996089-Em-que-cidades-opera-a-Cabify. [Accessed: 26-Nov-2017].

[37] Cabify, "Estimativa de Preços," 2018. [Online]. Available: https://cabify.com/es/portugal/porto#tariffs-lite. [Accessed: 27-Sep-2018].

[38] Cabify, "Cabify: Transporte de menores: é possível viajar sem sistemas de retenção para crianças?" [Online]. Available: https://help.cabify.com/hc/pt/articles/115001402909-Transporte-de-menores-é-possível-viajar-sem-sistemas-de-retenção-para-crianças-. [Accessed: 26-Nov-2017].

[39] Cabify, "Cabify: O que é o Cabify Express?" [Online]. Available: https://help.cabify.com/hc/pt/articles/115000825245-O-que-é-o-Cabify-Express-. [Accessed: 26-Nov-2017].

[40] Cabify, "Cabify: Os animais de estimação podem viajar na Cabify?" [Online]. Available: https://help.cabify.com/hc/pt-br/articles/115001090445-Os-animais-de-estimação-podem-viajar-na-Cabify. [Accessed: 26-Nov-2017].

[41] QC Veículos, "Lyft: Conheça o principal concorrente do Uber." [Online]. Available: http://qcveiculos.com.br/lyft-conheca-o-principal-concorrente-uber/. [Accessed: 07-Dec-2017].

[42] CNET Networks Incorporated, "Uber vs Lyft: 9 things to consider before your first ride," 2015. [Online]. Available: https://www.cnet.com/how-to/uber-lyft-ride-share-ride-hailing/. [Accessed: 06-Dec-2017].

[43] Fortune, "Lyft Just Made Its Biggest One-Day Expansion Into U.S. Cities," 2017. [Online]. Available: http://fortune.com/2017/02/23/lyft-54-cities/. [Accessed: 07-Dec-2017].

[44] TechCrunch, "Lyft's first market outside the U.S. will be Canada with a December launch in Toronto," 2017. [Online]. Available: https://techcrunch.com/2017/11/13/lyfts-first-market-outside-the-u-s-will-be-canada-with-a-december-launch-in-toronto/. [Accessed: 07-Dec-2017].

[45] Lyft Estimate, "Where is Lyft Available." [Online]. Available: https://lyftrideestimate.com/available/lisboa-pt. [Accessed: 06-Dec-2017].

[46] Aborrecido, "O que é Lyft?" [Online]. Available: http://aborrecido.ru/casa-e-jardim/como-viver-verde/transporte/31646-o-que-lyft.html. [Accessed: 07-Dec-2017].

[47] Jornal de Negócios, "BlaBlaCar: Dar boleia para conquistar a Europa," 2014. [Online]. Available: http://www.jornaldenegocios.pt/empresas/pme/start-ups/detalhe/blablacar_dar_boleia_para_conquistar_a_europa. [Accessed: 28-Nov-2017].

[48] BlaBlaCar, "A nossa história." [Online]. Available: https://www.blablacar.pt/sobre-a-blablacar/a-nossa-historia. [Accessed: 28-Nov-2017].

[49] BlaBlaCar, "O que é a BlaBlaCar?" [Online]. Available: https://www.blablacar.pt/faq/pergunta/o-que-e-a-blablacar. [Accessed: 28-Nov-2017].

[50] BlaBlaCar, "Quais são as vantages de usar a BlaBlaCar?" [Online]. Available: https://www.blablacar.pt/faq/pergunta/o-que-tem-a-blablacar-para-mim. [Accessed: 30-Nov-2017].

[51] BlaBlaCar, "BlaBlaCar: Página Inicial." [Online]. Available: https://www.blablacar.pt. [Accessed: 28-Nov-2017].

[52] BlaBlaCar, "Como funciona a BlaBlaCar?" [Online]. Available: https://www.blablacar.pt/faq/pergunta/como-funciona-a-blablacar. [Accessed: 30-Nov-2017].

[53] BlaBlaCar, "É necessários estar registado?" [Online]. Available: https://www.blablacar.pt/faq/pergunta/porque-tenho-de-registar-me-e-porque-e-gratuito. [Accessed: 30-Nov-2017].

[54] BlaBlaCar, "Como o preço é definido?" [Online]. Available: https://www.blablacar.pt/faq/pergunta/como-funciona-o-preco-e-o-metodo-de-pagamento. [Accessed: 30-Nov-2017].

[55] BlaBlaCar, "Não consigo pagar com o meu cartão. O que faço?" [Online]. Available: https://www.blablacar.pt/faq/pergunta/no-consigo-pagar-com-o-meu-carto-o-que-fao. [Accessed: 30-Nov-2017].

[56] BlaBlaCar, "E se não tiver cartão de crédito?" [Online]. Available: https://www.blablacar.pt/faq/pergunta/e-se-no-tiver-carto-de-crdito. [Accessed: 30-Nov-2017].

[57] BlaBlaCar, "Como pagar as minhas reservas utilizando a minha conta PayPal?" [Online]. Available: https://www.blablacar.pt/faq/pergunta/como-pagar-as-minhas-reservas-utilizando-a-minha-conta-paypal. [Accessed: 30-Nov-2017].

[58] 360meridianos, "BlaBlaCar: Dicas para usar o aplicativo de carona e economizar na viagem," 2017. [Online]. Available: https://www.360meridianos.com/2017/02/blablacar-aplicativo-de-carona-dicas.html. [Accessed: 30-Nov-2017].

[59] Turista imPerfeito, "BlaBlaCar: Tudo o que você precisa de saber," 2015. [Online]. Available: https://www.turistaimperfeito.com/blablacar-tudo-o-que-voce-precisa-saber/. [Accessed: 30-Nov-2017].

[60] BlaBlaCar, "BlaBlaCar Developer Portal: API Documentation." [Online]. Available: https://dev.blablacar.com/docs/versions/1.0/getting-started. [Accessed: 01-Dec-2017].

[61] Meio & Mensagem, "99Taxis muda de nome e lança vídeo manifesto," 2016. [Online]. Available: http://www.meioemensagem.com.br/home/marketing/2016/07/07/99-taxis-muda-de-nome-e-lanca-video-manifesto.html.

[62] J. Morgado, "Criador do aplicativo 99Taxis conta a história de seu empreendimento.," 2017. [Online]. Available: https://pt.linkedin.com/pulse/criador-do-aplicativo-99taxis-conta-história-de-seu-jorge-morgado. [Accessed: 07-Dec-2017].

[63] SAPOTek, "Aplicação 99Taxis vem à descoberta de Portugal," 2014. [Online]. Available: windows/artigos/aplicacao-99taxis-vem-a-descoberta-de-portugal. [Accessed: 29-Dec-2017].

[64] Pplware, "99Taxis ultrapassa 800 taxistas em Lisboa e entra no Porto," 2015. [Online]. Available: https://pplware.sapo.pt/smartphones-tablets/android/99taxis-ultrapassa-800-taxistas-em-lisboa-e-entra-no-porto/. [Accessed: 29-Dec-2017].

[65] 99, "Qual a diferença entre as categorias da 99?" [Online]. Available: https://99novo.zendesk.com/hc/pt-br/articles/115014402868-Qual-a-diferença-entre-as-

categorias-da-99-. [Accessed: 29-Dec-2017].
[66] Ambitur, "Chegou a MyTaxi, a app que fará frente à Uber em Lisboa." [Online]. Available: http://www.ambitur.pt/chegou-a-mytaxi-a-app-que-fara-frente-a-uber-em-lisboa/. [Accessed: 29-Dec-2017].
[67] Intelligent Apps GmbH, "MyTaxi: Acerca da MyTaxi." [Online]. Available: https://pt.mytaxi.com/jobs/acerca-da-mytaxi.html?erid=1514506392776604102. [Accessed: 29-Dec-2017].
[68] Pplware, "mytaxi: Nunca foi tão simples andar de táxi," 2016. [Online]. Available: https://pplware.sapo.pt/smartphones-tablets/mytaxi-nunca-simples-andar-taxi/. [Accessed: 29-Dec-2017].
[69] Shifter, "Com quantas apps podemos ir de A a B?," 2017. [Online]. Available: https://shifter.pt/2017/04/codigo-cabify-viagem-gratis/. [Accessed: 29-Dec-2017].
[70] Ambitur, "Thumbeo: Uma app de partilha de boleias," 2017. [Online]. Available: http://www.ambitur.pt/thumbeo-uma-app-de-partilha-de-boleias/. [Accessed: 21-Dec-2017].
[71] Thumbeo, "Thumbeo: Página Inicial." [Online]. Available: http://www.thumbeo.com. [Accessed: 21-Dec-2017].
[72] Público, "Thumbeo, a app de 'boleias personalizadas' criada por dois universitários," 2015. [Online]. Available: http://p3.publico.pt/vicios/hightech/17425/thumbeo-app-de-boleias-personalizadas-criada-por-dois-universitarios. [Accessed: 21-Dec-2017].
[73] Talkfest, "Shortlists Iberian Festival Awards," 2016. [Online]. Available: https://www.talkfest.eu/single-post/2016/02/11/Shortlists-Iberian-Festival-Awards. [Accessed: 21-Dec-2017].
[74] NiT, "Testámos o Citydrive: vale a pena ter dois carros?," 2015. [Online]. Available: https://nit.pt/out-of-town/12-07-2015-testamos-o-citydrive-vale-a-pena-ter-dois-carros. [Accessed: 03-Dec-2017].
[75] Fleet Magazine, "Citydrive com nova aplicação," 2017. [Online]. Available: http://fleetmagazine.pt/2017/07/21/citydrive-nova-aplicacao/. [Accessed: 03-Dec-2017].
[76] DriveNow S.A., "DriveNow: Resumo dos preços DriveNow." [Online]. Available: https://www.drive-now.com/pt/pt/pricing/. [Accessed: 03-Dec-2017].
[77] DriveNow S.A., "Como funciona a DriveNow?" [Online]. Available: https://www.drive-now.com/pt/pt/how-it-works. [Accessed: 03-Dec-2017].
[78] PME Magazine, "Conheça a plataforma de carsharing feita em Portugal." [Online]. Available: https://pmemagazine.com/conheca-a-plataforma-de-carsharing-feita-em-portugal/. [Accessed: 03-Dec-2017].
[79] BookingDrive, "BookingDrive: Criar WebApp Bookingdrive.com no seu telemóvel." [Online]. Available: https://www.bookingdrive.com/staticpage/slugview?slug=webapp. [Accessed: 03-Dec-2017].
[80] BookingDrive, "BookingDrive: Quais são as formas de pagamento aceites?" [Online]. Available: https://www.bookingdrive.com/page/faq#questoesgerais-2. [Accessed: 03-Dec-2017].
[81] SIC Notícias, "Bookingdrive chega em dezembro a Portugal." [Online]. Available: http://sicnoticias.sapo.pt/economia/2016-11-05-Bookingdrive-chega-em-dezembro-a-Portugal. [Accessed: 03-Dec-2017].
[82] BookingDrive, "BookingDrive: Quando é que o contrato de aluguer é assinado?" [Online]. Available: https://www.bookingdrive.com/page/faq#questoesgerais-5. [Accessed: 03-Dec-2017].
[83] Unleashed LLC, "Lista de Perguntas Frequentes," 2018. [Online]. Available: https://www.taxifarefinder.com/faq.php. [Accessed: 25-Oct-2018].
[84] Unleashed LLC, "TaxiFareFinder API," 2018. [Online]. Available: http://webcache.googleusercontent.com/search?q=cache:https://www.taxifarefinder.com/api.php&strip=0&vwsrc=0. [Accessed: 25-Oct-2018].
[85] A. Hartmans, "Google just unveiled a feature that Uber has been blocking startups from making for years," 2016. [Online]. Available: https://www.businessinsider.com/google-ride-sharing-price-comparison-tool-2016-9. [Accessed: 24-Aug-2018].
[86] Bellhop Technologies Inc, "Bellhop - Compare Rideshares," 2018. [Online]. Available: https://itunes.apple.com/pt/app/bellhop-compare-rideshares/id1081008679?mt=8. [Accessed: 28-Aug-2018].

[87] PWM, "O que é um Crawler," 2016. [Online]. Available: https://www.pwm.pt/Webmarketing/SEO/OqueéumCrawler/tabid/4729/Default.aspx. [Accessed: 30-Jul-2018].

[88] A. Chang, "The Facebook and Cambridge Analytica scandal, explained with a simple diagram," 2018. [Online]. Available: https://www.vox.com/policy-and-politics/2018/3/23/17151916/facebook-cambridge-analytica-trump-diagram. [Accessed: 08-Oct-2018].

[89] I. Sherr, "Facebook, Cambridge Analytica and data mining: What you need to know," 2018. .

[90] José Luís Caldeira Pinto, "API RESTful para Sistema de Gestão de Desempenho de Redes de Telecomunicações," Universidade de Aveiro, 2015.

[91] W. Santos, "Research Shows Interest in Providing APIs Still High," 2018. [Online]. Available: https://www.programmableweb.com/news/research-shows-interest-providing-apis-still-high/research/2018/02/23. [Accessed: 20-Aug-2018].

[92] M. Endrei *et al.*, *Patterns: Service-Oriented Architecture and Web Services*, vol. 1. 2004.

[93] Pedro Jorge Pereira Lopes, "Service composition for biomedical applications," Universidade de Aveiro, 2012.

[94] I. S. Kang, R. Morais, and W. Santos, "Padrão TISS: Um Estudo de Caso sobre o uso de web services para o seu desenvolvimento," *12th CONTECSI Int. Conf. Inf. Syst. Technol. Manag.*, pp. 1–17, 2015.

[95] A. Marques, "Desenvolvimento de API para aplicação cloud," Instituto Politécnico de Leiria, 2018.

[96] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, 2000.

[97] NordicAPIs, "SOAP VS REST - A NordicAPIs infographic." [Online]. Available: https://nordicapis.com/rest-vs-soap-nordic-apis-infographic-comparison/. [Accessed: 18-Oct-2018].

[98] F. P. de Souza, "Criação de framework REST/HATEOAS Open Source para desenvolvimento de APIs em Node.js," Faculdade de Engenharia Da Universidade do Porto, 2015.

[99] "REST Architectural Constraints." [Online]. Available: https://restfulapi.net/rest-architectural-constraints/. [Accessed: 08-Nov-2018].

[100] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing," *Internet Eng. Task Force*, 2014.

[101] Wikipédia, "Hypertext Transfer Protocol," 2018. [Online]. Available: https://pt.wikipedia.org/wiki/Hypertext_Transfer_Protocol. [Accessed: 26-Jun-2018].

[102] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," *Internet Eng. Task Force*, pp. 1–101, 2014.

[103] Django Software Foundation, "Why Django?" [Online]. Available: https://www.djangoproject.com/start/overview. [Accessed: 29-Jun-2018].

[104] C. Rocha, "Padrões Arquiteturais MVC X Arquitetura do Django," 2016. [Online]. Available: https://github.com/fga-gpp-mds/A-Disciplina/wiki/Padrões-Arquiteturais---MVC-X-Arquitetura-do-Django. [Accessed: 29-Jun-2018].

[105] DevMedia, "ORM : Object Relational Mapper," 2011. [Online]. Available: https://www.devmedia.com.br/orm-object-relational-mapper/19056. [Accessed: 18-Oct-2018].

[106] Full Stack Python, "Object-relational mappers (ORMs)," 2018. [Online]. Available: https://www.fullstackpython.com/object-relational-mappers-orms.html. [Accessed: 18-Oct-2018].

[107] A. Holmes, "Reviewing Django REST Framework," 2014. [Online]. Available: https://www.isotoma.com/blog/2014/03/25/reviewing-django-rest-framework/. [Accessed: 09-Jul-2018].

[108] H. Brendon, "Tarefas assíncronas no Django com Celery," 2017. [Online]. Available: https://medium.com/@hudsonbrendon/tarefas-assíncronas-no-django-com-celery-1-3-dd73c1bb92f5. [Accessed: 13-Aug-2018].

[109] F. Alves, "Tarefas demoradas de forma assíncrona com Django e Celery," 2015. [Online]. Available: https://fernandofreitasalves.com/tarefas-assincronas-com-django-e-celery/. [Accessed: 13-Aug-2018].

[110] SQLite, "SQL As Understood By SQLite." [Online]. Available: https://sqlite.org/lang.html.

[Accessed: 23-Oct-2018].
[111] SQLite, "SQLite Is A Zero-Configuration Database." [Online]. Available: https://sqlite.org/zeroconf.html. [Accessed: 23-Oct-2018].
[112] Django Software Foundation, "Django Models," 2018. [Online]. Available: https://docs.djangoproject.com/en/1.11/topics/db/models/. [Accessed: 21-Aug-2018].
[113] Django Software Foundation, "Model field reference," 2018. [Online]. Available: https://docs.djangoproject.com/en/2.1/ref/models/fields/#field-types. [Accessed: 21-Aug-2018].
[114] Uber Technologies Inc., "Github: Rides SDK." [Online]. Available: https://github.com/uber/rides-python-sdk.
[115] Wikipedia, "Regular expression," 2018. [Online]. Available: https://en.wikipedia.org/wiki/Regular_expression. [Accessed: 29-Jul-2018].
[116] Google, "Google JSON Style Guide." [Online]. Available: https://google.github.io/styleguide/jsoncstyleguide.xml. [Accessed: 31-Aug-2018].
[117] Standard C++ Foundation, "Serialization and Unserialization," 2018. [Online]. Available: https://isocpp.org/wiki/faq/serialization. [Accessed: 11-Jul-2018].
[118] M. Sporny, "JSON-LD 1.1 A JSON-based Serialization for Linked Data," 2018. [Online]. Available: https://json-ld.org/spec/latest/json-ld/. [Accessed: 12-Jul-2018].
[119] A. Oliveira and F. Firmo, "REST - padrões e melhores práticas," 2012. [Online]. Available: https://pt.slideshare.net/Sensedia/rest-padroes-e-melhores-praticas?from_action=save. [Accessed: 10-Jul-2018].
[120] T. Sokol, "Tips and tricks for API pagination," 2017. [Online]. Available: https://medium.com/square-corner-blog/tips-and-tricks-for-api-pagination-5cacc6f017da. [Accessed: 10-Jul-2018].
[121] Atlassian, "Pagination in the REST API," 2018. [Online]. Available: https://developer.atlassian.com/server/confluence/pagination-in-the-rest-api/. [Accessed: 10-Jul-2018].
[122] Django REST Framework, "Pagination: CursorPagination." [Online]. Available: http://www.django-rest-framework.org/api-guide/pagination/#cursorpagination. [Accessed: 18-Jul-2018].
[123] Science Buddies, "The Engineering Design Process," 2018. [Online]. Available: https://www.sciencebuddies.org/science-fair-projects/engineering-design-process/engineering-design-process-steps. [Accessed: 12-Nov-2018].
[124] M. Saini and K. Kaur, "A Review of Open Source Software Development Life Cycle Models," *Int. J. Softw. Eng. Its Appl.*, vol. 8, no. 3, pp. 417–434, 2014.
[125] R. P. dos S. Oliveira, "Sistema de monitorização e controlo da produção de Salicornia na Ria de Aveiro," Universidade de Aveiro, 2017.
[126] The Apache Software Foundation, "Apache JMeter™," 2018. [Online]. Available: https://jmeter.apache.org. [Accessed: 30-Oct-2018].
[127] Django Rest Framework, "Django Rest Framework," 2018. [Online]. Available: https://www.django-rest-framework.org. [Accessed: 14-Nov-2018].

ANEXO **A**

Documentação da API

As figuras apresentadas de seguida são referentes à documentação gerada a todos os métodos de cada *endpoint*, a partir da framework Swagger. Esta ferramenta *open-source* permite gerar, de um modo semi-automático, a documentação de soluções desenvolvidas, como é o caso de uma API REST.

De lembrar que todos os *endpoints*, à exceção do `/rides`, têm implementado os quatro métodos que permitem as quatro operações CRUD, são eles, o GET, DELETE, POST e PUT. Uma vez que o `/rides` têm como objetivo o retorno de uma coleção de viagens de todos os *providers* presentes na solução, os métodos POST e PUT não se mostram assertivos, uma vez que, não está explicito para que *provider* se deseja criar ou editar uma viagem. Deste modo, o `/rides` apenas tem implementado os métodos GET e DELETE.

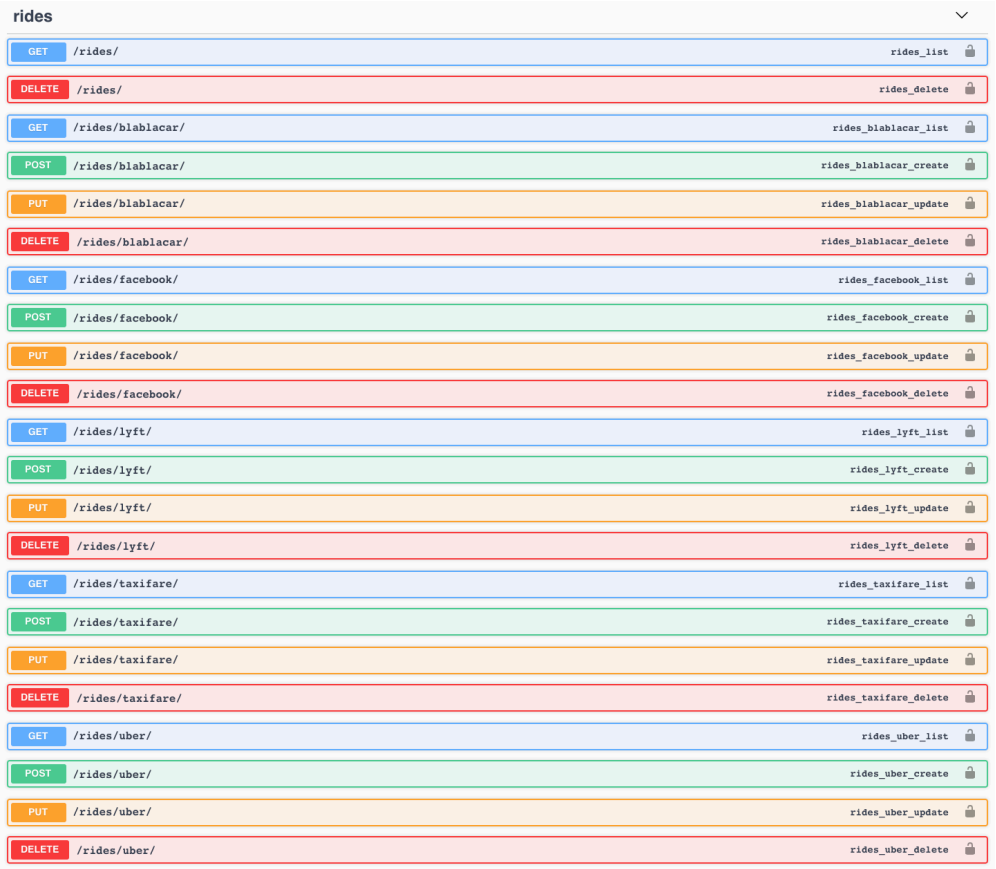
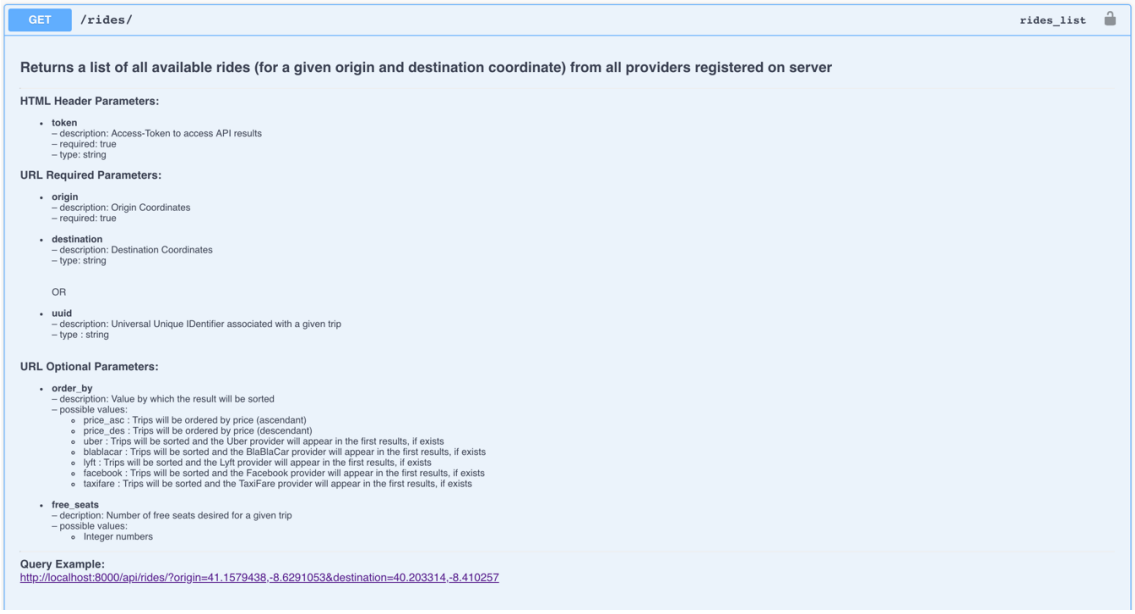


Figura 72 – Lista de métodos associada a cada endpoint

Métodos associados ao *endpoint* `/rides`

- **GET**



GET /rides/ rides_list

Returns a list of all available rides (for a given origin and destination coordinate) from all providers registered on server

HTML Header Parameters:

- **token**
  – description: Access-Token to access API results
  – required: true
  – type: string

URL Required Parameters:

- **origin**
  – description: Origin Coordinates
  – required: true
- **destination**
  – description: Destination Coordinates
  – type: string

OR

- **uuid**
  – description: Universal Unique IDentifier associated with a given trip
  – type : string

URL Optional Parameters:

- **order_by**
  – description: Value by which the result will be sorted
  – possible values:
    - price_asc : Trips will be ordered by price (ascendant)
    - price_des : Trips will be ordered by price (descendant)
    - uber : Trips will be sorted and the Uber provider will appear in the first results, if exists
    - blablacar : Trips will be sorted and the BlaBlaCar provider will appear in the first results, if exists
    - lyft : Trips will be sorted and the Lyft provider will appear in the first results, if exists
    - facebook : Trips will be sorted and the Facebook provider will appear in the first results, if exists
    - taxifare : Trips will be sorted and the TaxiFare provider will appear in the first results, if exists
- **free_seats**
  – decription: Number of free seats desired for a given trip
  – possible values:
    - Integer numbers

Query Example:
http://localhost:8000/api/rides/?origin=41.1579438,-8.6291053&destination=40.203314,-8.410257

Figura 73 – Descrição do método GET do /rides

Response Example:


```
{
    "current_page":1
    "total_results": 7,
    "total_pages":2,
    "next_page": "http://localhost:8000/api/rides/?destination=40.203314%2C-8.410257&origin=41.1579438%2C-8.6291053&page=2",
    "previous_page": null,
    "results": [
        {
            "origin": "41.1579438,-8.6291053",
            "destination": "40.203314,-8.410257",
            "origin_name": "Praça Mouzinho de Albuquerque (Heróis da Guerra Peninsular), 4149-071, Porto, Portugal",
            "destination_name": "Rua Júlio Dinis (Igreja de São José), 3030-775, Coimbra, Portugal",
            "trip_duration": 81,
            "trip_distance": 75.63,
            "trip_datetime": "2018-05-16T14:11:21+00:00",
            "price": 99.5,
            "price_currency" : "EUR",
            "free_seats": 4,
            "provider": "UBER",
            "provider_data": {
                "car_type": "uberX",
                "estimated_min_price": 86,
                "estimated_max_price": 113,
                "multiplier": 1,
                "trip_id": "4dba36f8-e760-42f6-8da4-5515aaac629b"
            },
            "uuid": "7eb787d4-e9e9-47fe-8b6e-b41fbc2e2bb0"
        },
        {
            "origin": "41.1579438,-8.6291053",
            "destination": "40.203314,-8.410257",
            "origin_name": "Porto - Campanha, Porto, Portugal",
            "destination_name": "Portugal dos Pequenitos, Largo Rossio de Santa Clara, Coimbra, Portugal",
            "trip_duration": 80,
            "trip_distance": 118,
            "trip_datetime": "2018-05-16T18:00:00+00",
            "price": 7,
            "free_seats": 3,
            "provider": "BLABLACAR",
            "provider_data": {
                "car_info": {
                    "id": "55683543",
                    "model": "308SW",
                    "make": "PEUGEOT",
                    "comfort": "Normal",
                    "comfort_nb_star": 2
                },
                "trip_id": "Cg4KCjEwOTc3MzM0NzEQARILCAISB2VGM2ZoZWgaCwgCEgdlejRoOG5k"
            },
            "uuid": "75fa7a6b-2446-47c8-94c2-c8b1f6951c73"
        }
    ]
}
```


Figura 74 – Exemplo de resposta do método GET ao /rides

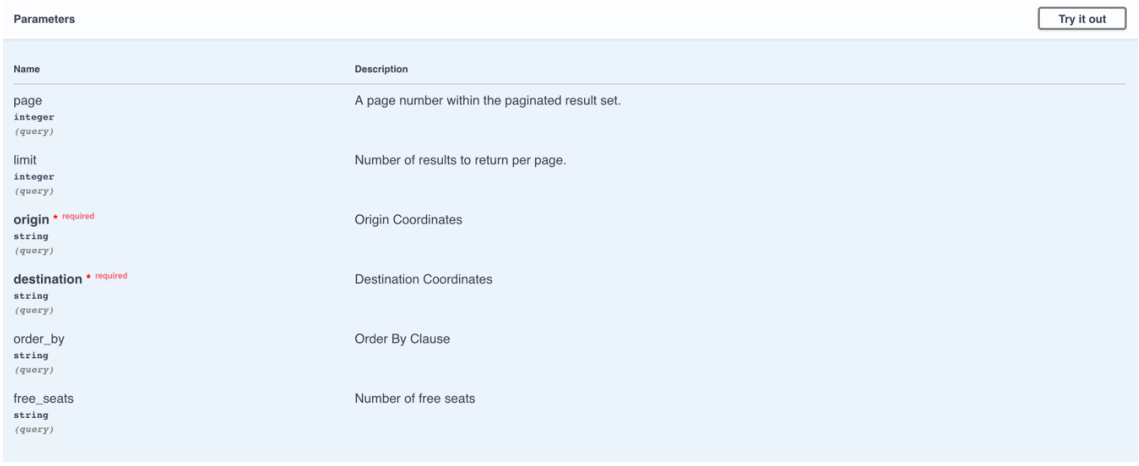


Figura 75 – Parâmetros obrigatórios e não-obrigatórios do método GET ao /rides

- **DELETE**

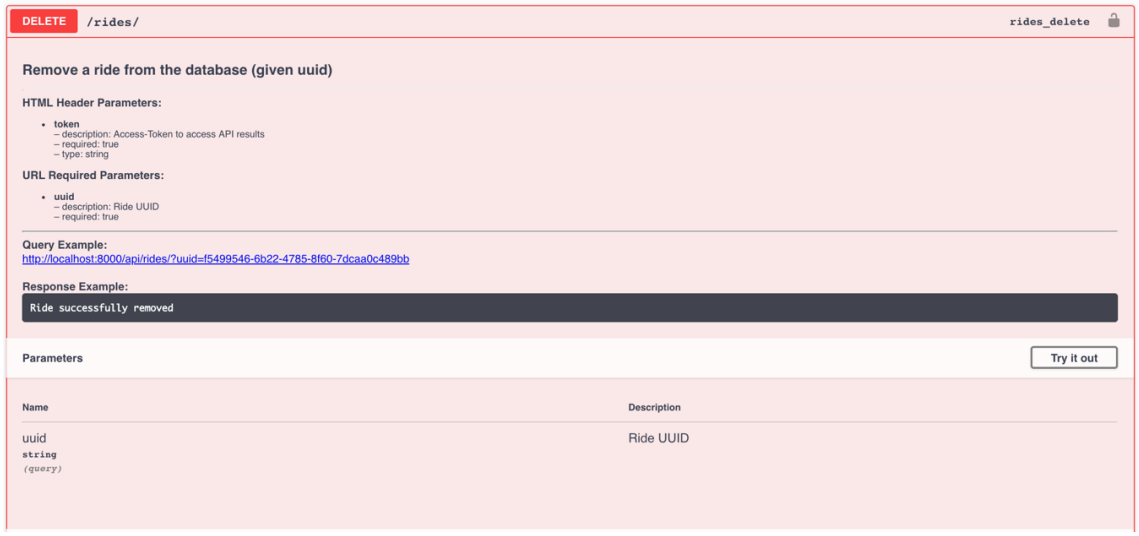


Figura 76 – Descrição, exemplo de resposta e parâmetros do método DELETE ao /rides

Métodos associados ao *endpoint* `/blablacar`

(análogo para os restantes *providers*, variando o conteúdo da resposta obtida)

- **GET**

Figura 77 – Descrição do método GET do /rides/blablacar

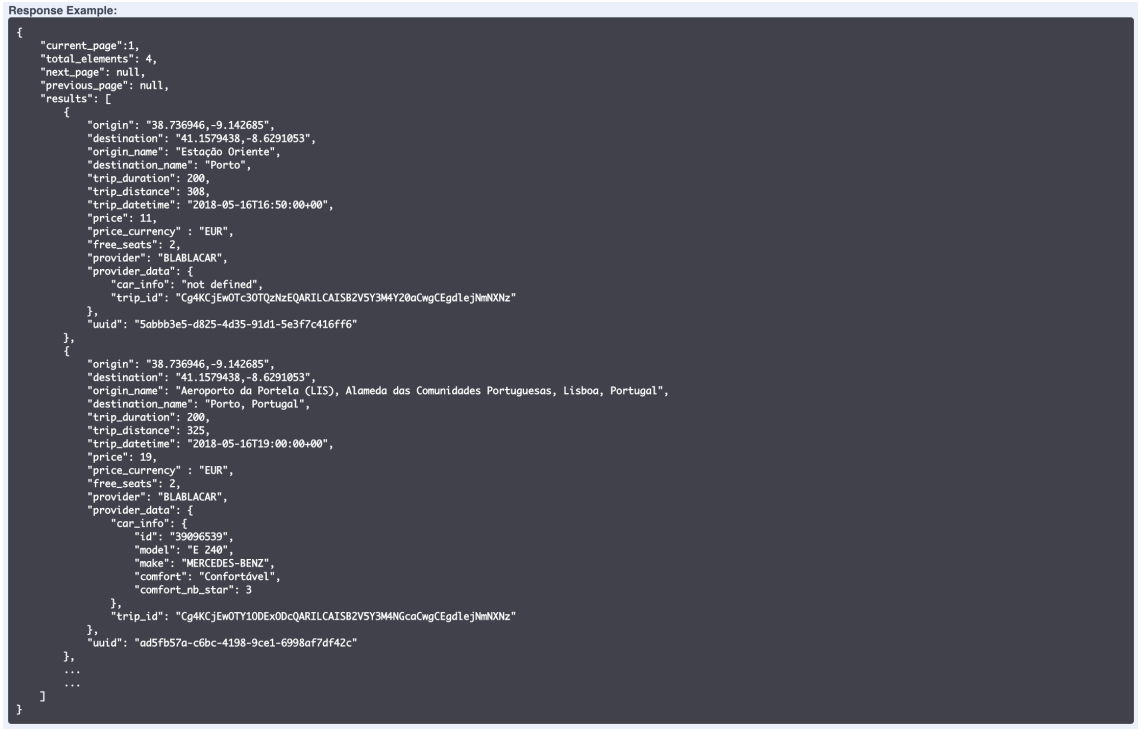


Figura 78 – Exemplo de resposta do método GET ao /rides/blablacar

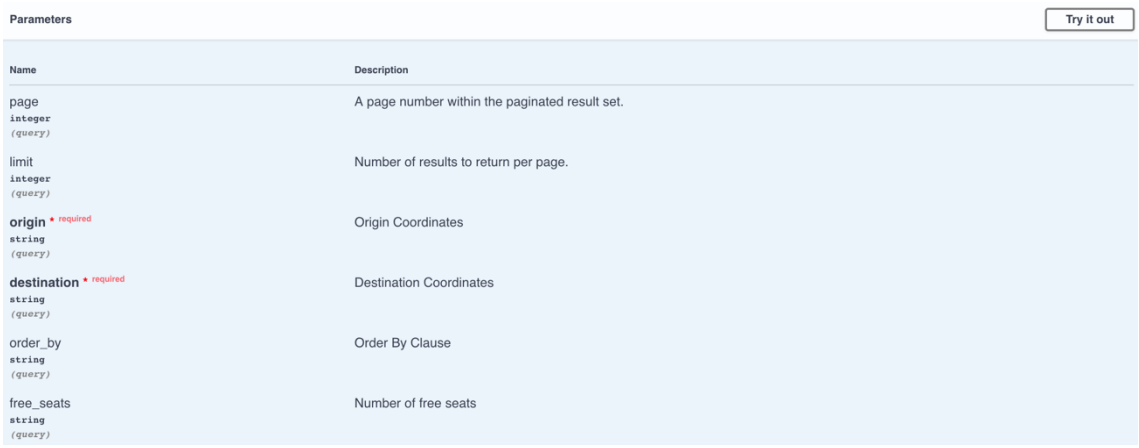


Figura 79 – Parâmetros obrigatórios e não-obrigatórios do método GET ao /rides/blablacar

- **DELETE**

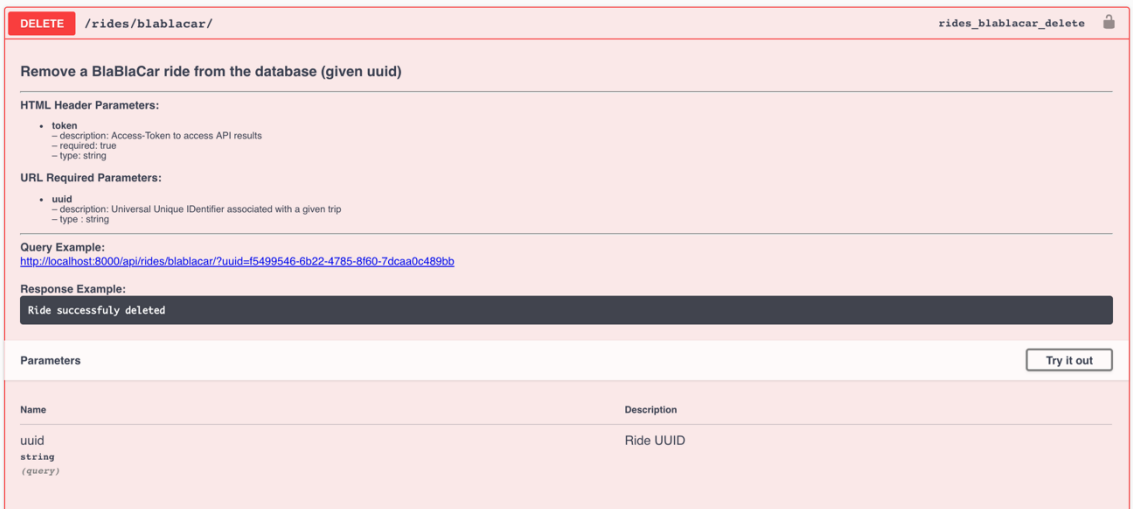


Figura 80 – Descrição, exemplo de resposta e parâmetros do método DELETE ao /rides/blablacar

- **POST**

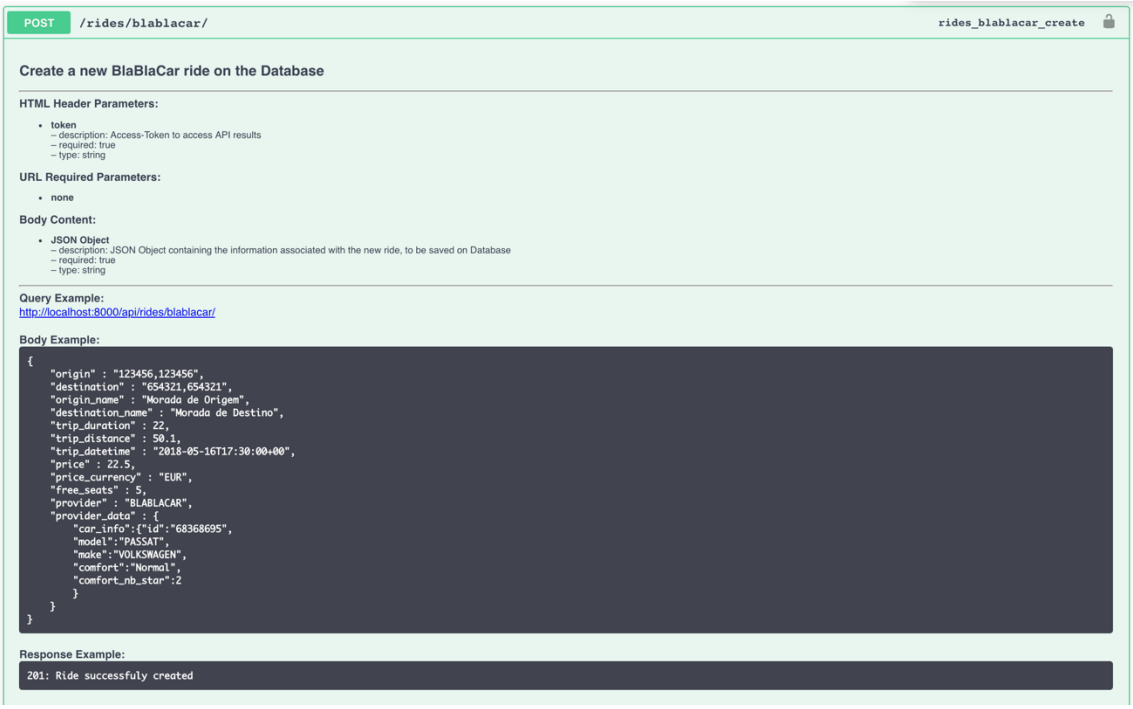


Figura 81 – Descrição e exemplo de resposta do método POST ao /rides/blablacar

- **PUT**

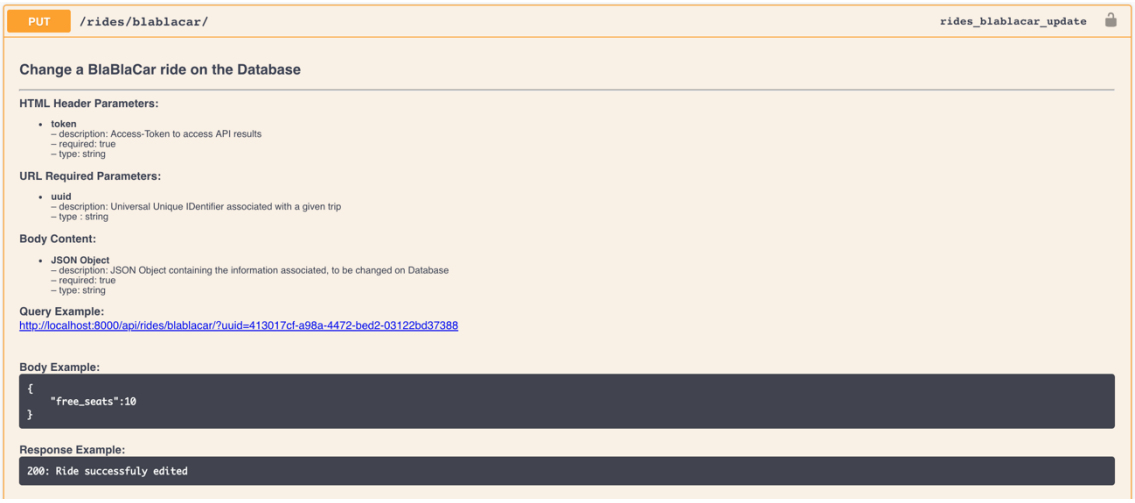


Figura 82 – Descrição, exemplo de resposta e parâmetros do método PUT ao /rides/blablacar